Embedded Artificial Intelligence for Tactile Sensing

Mohamad Alameh

Supervisor: Prof. Maurizio Valle

Co-Supervisor: Dr.Ali Ibrahim

DITEN - Dipartimento di Ingegneria Navale, Elettrica, Elettronica e delle Telecommunicazioni

Università degli studi di Genova

This dissertation is submitted for the degree of

Doctor of Philosophy

in

Science and Technology for Electronic and Telecommunication Engineering

Cycle XXXIII (2017-2020)

Coordinatorator: Prof. Mario Marchese

COSMIC Lab - DITEN February 2021

I would like to dedicate this thesis to:

My Mother & my Father, the reason of my existence...

The love of my life...

Who sacrifices everything to let me reach each and every high position...

My lovely and inspiring kids, who gives me all the motivation...

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Mohamad Alameh

February 2021

Acknowledgements

And I would like to acknowledge ...

My supervisors, Prof. Maurizio Valle and Dr. Ali Ibrahim...

Who were more than supervisors, but real teachers and leaders...

The coordinator of my Ph.D, Prof. Mario Marchese...

My colleagues who were always the support inside and outside the research life...

All UniGe staff, and silent workers who work always in the background...

The "Regione Liguria", who supported my right to study...

People who contributed to this work...

And everyone who gave me a push to accomplish this milestone, even with a smile...

Finally, thanks to all the reviewers and readers of this work.

Abstract

Electronic tactile sensing becomes an active research field whether for prosthetic applications, robotics, virtual reality or post stroke patients rehabilitation. To achieve such sensing, an array of sensors is used to retrieve human-skin like information, which is called Electronic skin (E-skin). Humans through their skins, are able to collect different types of information e.g. pressure, temperature, texture, etc. which are then passed to the nervous system, and finally to the brain in order to extract high level information from these sensory data. In order to make E-skin capable of such task, data acquired from E-skin should be filtered, processed, and then conveyed to the user (or robot). Processing these sensory information, should occur in real-time, taking in consideration the power limitation in such applications, especially prosthetic applications. The power consumption itself is related to different factors, one factor is the complexity of the algorithm e.g. number of FLOPs, and another is the memory consumption.

In this thesis, I will focus on the processing of real tactile information, by 1)exploring different algorithms and methods for tactile data classification, 2)data organization and preprocessing of such tactile data and 3)hardware implementation. More precisely the focus will be on deep learning algorithms for tactile data processing mainly CNNs and RNNs, with energy-efficient embedded implementations.

The proposed solution has proved less memory, FLOPs, and latency compared to the state of art (including tensorial SVM), applied to real tactile sensors data.

Keywords: E-skin, tactile data processing, deep learning, CNN, RNN, LSTM, GRU, embedded, energy-efficient algorithms, edge computing, artificial intelligence.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Tactile sensing systems attract the research interest in many application domains such as robotics, prosthetic devices, and industrial automation [1, 2]. The main focus is in the areas of sensors and transducers, front end electronics, and smart data processing algorithms. In this chapter, we will give a real example of a tactile sensing system realised at COSMIC Lab, where we explore the different components, and to highlight data processing part, subject of this thesis.

1.1 Electronic Skin System

n electronic skin system (E-Skin) is composed of: 1) an array of tactile sensors to sense the applied mechanical stimuli, 2) an interface electronics for signal conditioning and data acquisition, and 3) an embedded digital processing unit for tactile data decoding. The goal is either to mimic the human capabilities in capturing and interpreting tactile data or to respond to the application demands. To be effective, tactile sensors have to sense and extract meaningful information from the contact surface such as force direction and intensity, position, vibrations, objects and texture, or touch modality classification. Such information can be extracted by employing algorithms rooted in machine learning, which have proven their effectiveness in processing tactile data. This processing needs to be real-time, while

taking into consideration the power limitation in portable/wearable applications where no continuous power sources are available e.g. prosthetics, Internet of things and mobile robots. In order to achieve such real-time computation, two solutions are available, with their own pros and cons:
1)To make near sensor processing, which means less latency but requires attention for the power consumption, which is nowadays known as Edge Computing [3].
2)Sending raw data to a remote computing unit which has higher computing capabilities, and continuous power source, but the latency will be higher, and attention should be taken to the bandwidth.

In this scope we will explore a tactile sensing system presented in two versions, in [4] where remote processing is employed, explained in subsection 1.1.1 , while in [5, 6] near-sensor processing is employed as explained in subsection 1.1.2.

1.1.1 Remote Computing E-Skin

This electronic skin system is composed of :

- Tactile sensors: capacitive touch array sensor, which can deliver up to 13×9 positions, through 22 electrodes.
- Interface electronics: the role here was to acquire data from sensors and send them to the PC via USB interface for visualization, and then sending necessary command to an electro-tactile stimulator, to convey the information to an amputee in a non invasive way. A commercial capacitive touch controller is used, with an STM32 micro-controller board, connected together through an I2C interface.
- Processing algorithm: to collect the acquired data, process them, visualize them on a graphical user interface, and finally send data to a stimulator via Bluetooth.

- Electro-tactile Stimulator: The stimulator role is to send a stimulation to the user according to the touch occurred in the tactile skin.

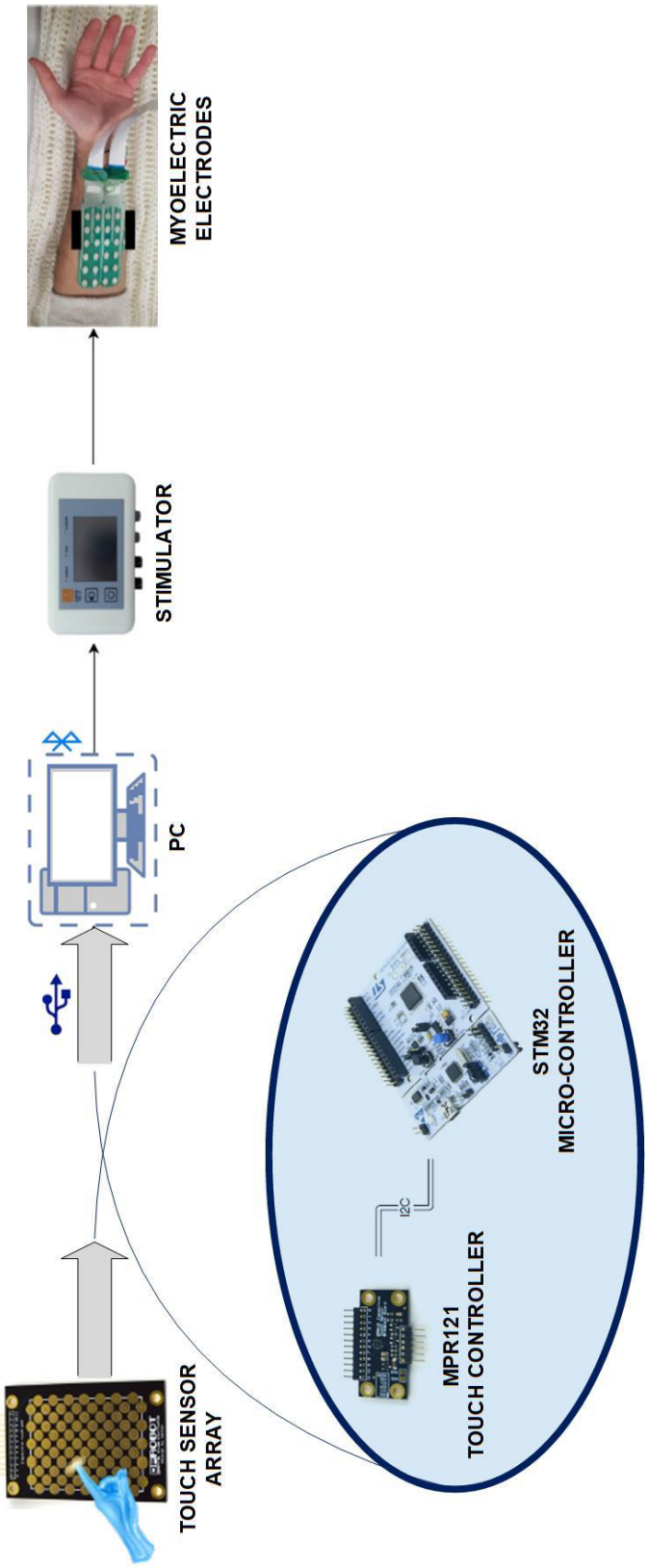


Fig. 1.1 Example of an E-skin System , with stimulation and remote processing

As shown in Figure 1.1, from the left a capactive sensory array, connected to an MPR121 touch controller, raw data are sent to an STM32 Micro-controller connected to PC via USB, then a software on the PC reads the data to visualize them on a Graphical User Interface (GUI), and send stimulation commands to the stimulator via an external Bluetooth interface. A snapshot of the graphical user interface is shown in Figure 1.2.

Codes used for the micro-controller and PC are listed in the Appendix B.

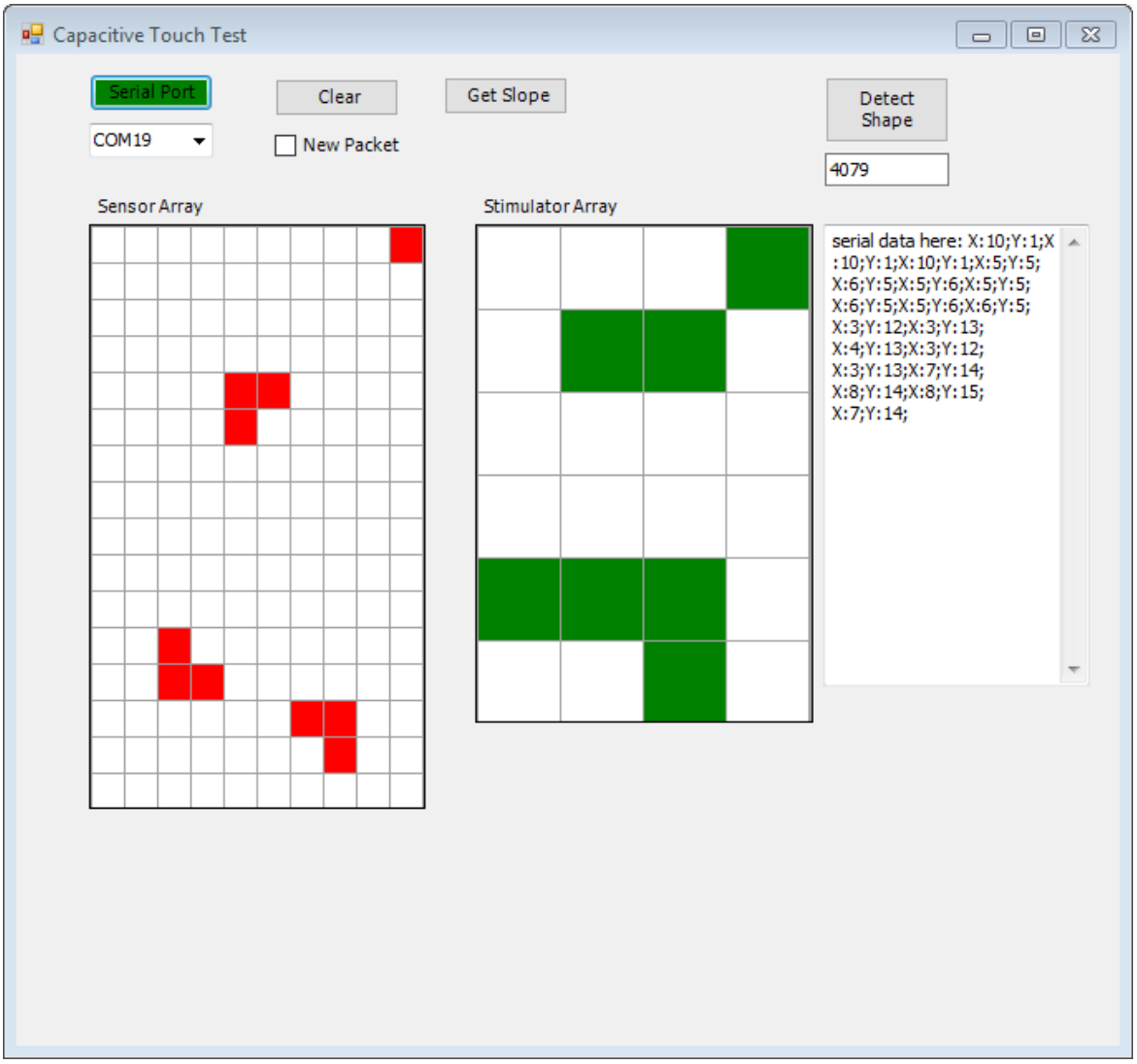


Fig. 1.2 Graphical user interface used for tactile data visualization, and sending commands to the stimulator

1.1.2 Edge Computing E-Skin

Another version of the previously mentioned system was reproduced by cancelling the PC, and making all the processing on the micro-controller side, then sending them to an electro-tactile stimulator via a Bluetooth interface

As shown in Figure 1.3, this system consists of:

- Tactile sensors: two different sensor arrays were tested: a)commercial 16×10 Force Sensitive Resistors (FSR), b) 4×4 Piezo-electric sensors, designed at our laboratory (COSMIC Lab) [7] .
- Interface electronics: A custom design interface electronics, equipped with a DDC232 for digital to analog conversion, and has a built-in Bluetooth interface.
- Processing algorithm: to collect the acquired data, process them, and finally send data to a stimulator via Bluetooth. The algorithm was implemented on the interface's MCU.
- Electro-tactile Stimulator: The stimulator role is to send a stimulation to the user according to the touch occurred in the tactile skin.

The processing of data in these two systems was simple, such as: detecting shapes of objects e.g. two shaped with two different sizes, and the direction of touch e.g. when sliding an object in straight line on the array, and then mapping them to a channel in the stimulator as mentioned in Figure 1.2. In the next chapters, we will explore more complex tactile data processing, and higher complexity algorithms which requires higher computing capabilities and therefore more power consumption, which is critical in portable applications.

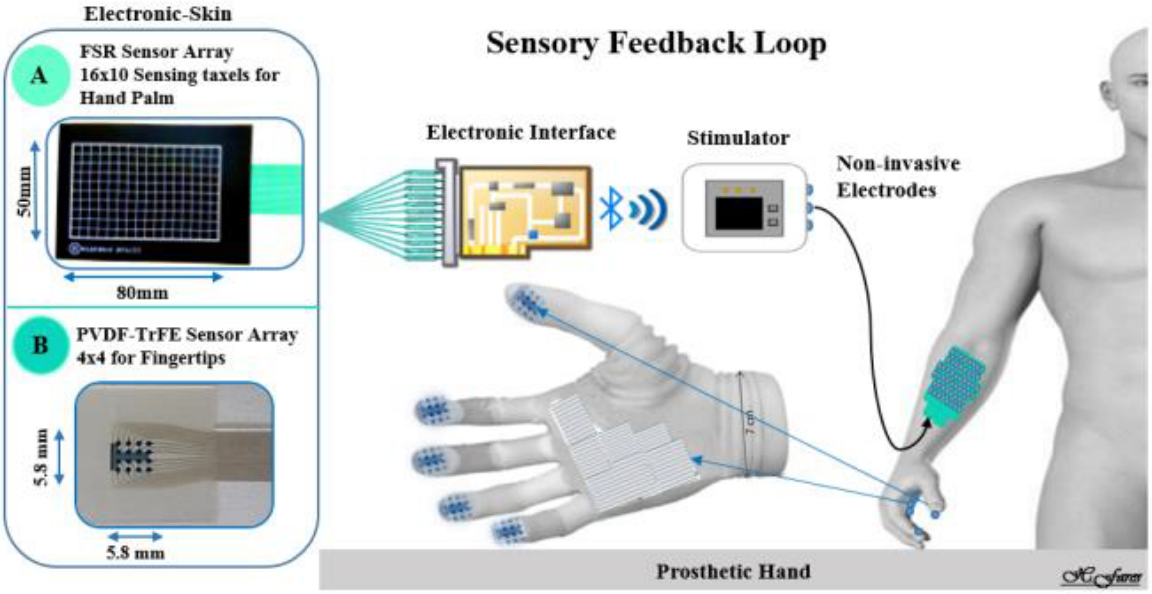


Fig. 1.3 Example of an E-skin System , with stimulation and on-edge processing

1.2 Artificial Intelligence

Artificial Intelligence (AI), invaded commercial and research fields in order to make machines solve in a human-like way different problems [8].

A definition of intelligent agent from the literature: any device that perceives its environment and takes actions that maximize its chance of successfully achieving its goals [9]. Colloquially, the term "artificial intelligence" is often used to describe machines (or computers) that mimic "cognitive" functions that humans associate with the human mind, such as "learning" and "problem solving" [8].

AI in brief is how a machine - especially a computing machine - is able to learn and solve a problem, even without having the model; conventional programming was based on rules and attributes were results are deterministic, i.e. for the same input you always have the same output for the same problem, and for a single problem you have one and only one solution.

While in artificial intelligence, you can have different models for solving the same problem, all solutions can be considered a good solution, they can differ by their complexity, their accuracy (when comparing the ground truth with the response of an AI system), or even

their structure. One important question here, is there the "Best" solution? There are good solutions, and solutions that are better than others concerning the metrics on which they are compared, but giving "the best" solution using AI, is not trivial. Unless you have an exact model, and in this way you are not doing machine learning, instead you are dealing with a well-defined model.

AI includes both machine learning and artificial neural networks(ANN). As an example of machine learning algorithms, Support Vector Machine (SVM) [10], K-Nearest-Neighbor (KNN) [11, 12], Random Forest [13], Decision Tree [14] etc. Artificial Neural Networks, or simply Neural Networks (NN) as a sub-category of machine learning, are a set of interconnected neurons, where connections have weights, and each neuron has an activation function. In order to train these networks, i.e. to make them solve a problem by giving an output similar to the expected, the weights should be modified through a training process [15, 16].

Machine learning tasks can be classified in two main categories, regression and classification, in regression the output is a continuous value e.g. estimating a price of a currency in the next day, while in classification the output is discreet and defined in a set value, to discriminate that an example belongs to one or more categories. The learning can be supervised or unsupervised, in an unsupervised learning, the role of the ML model is to make a clustering of the data, because the dataset is not labeled i.e. for each X we do not have a label Y, while in supervised learning, each sample of data is labeled, e.g. belongs to a certain group called a class. Different architectures are used in NNs as explained in [17]. A widely used NNs are Convolutional Neural Networks (CNNs) [18] used mainly in image processing, also some neural networks are designed for time-series-data processing (RNNs) [19].

Nowadays, there are other types of NN, which can generate new data similar to previously seen data, e.g. Generative Adversarial Networks (GAN). An example of a GAN network, is to generate new faces having seen a dataset of faces [20], or even merging a drawing style with a camera photo, in order to generate a painting from that photo, similar to the drawing

style [21]. Even neural networks are used to explore the best network architecture to be used for a solution [22].

1.2.1 Challenges and Constraints

There are a set of challenges for implementing AI algorithms on embedded platforms, for which new solutions arise daily, which make them accessible even at a commercial level, not only in research.

A. Latency

Latency is defined as the time difference between the generated output data and the input data provided to the system. In [23], [24] and [25] ML/DL algorithms take more than 1 second to classify different objects. This fact highlights the latency problem faced in IoT devices when implementing ML/DL algorithms; since the application must meet real time constraints.

B. Memory

Memory can be divided into two main categories, the program memory itself, which the size occupied by the algorithms instructions and its variables, and the input/output memory for data to be processed by that algorithm and the expected output respectively. The program memory can be very high especially in the case of Deep Learning, for storing the trainable parameters or even in some machine learning algorithms like the support vectors in Support Vector Machine (SVM). However, occupying large program memory induces a high amount of operations during computation due to the frequent transfer data between processors and different memory levels (on-chip , off-chip). For example more than 900 M operations of memory read and writes are needed in [26].

C. Complexity

By complexity we mean the number of floating point operations (FLOPs) needed in order to execute an algorithm, basically computed based on a single core architec-

ture. Moreover, using parallelism and multi-core technologies may help optimise the execution of complex algorithms.

A. Power/Energy consumption

Energy consumption is the power consumed during the execution of the algorithm when targeting embedded implementations ($J = W \times t$) where W is the power in Watt, t is the time in seconds, and J is energy in $Joules$. Energy efficiency is considered as an important metric especially when dealing with applications such as portable, medical/biomedical IoT devices. To emphasize the critical need of this metric, we take an example of the implemented tensorial SVM on FPGA device for classifying different touch modalities as shown in [27]. The proposed implementation is feasible for real time classification while the amount of power consumed is 1.14 W. Similarly as shown in [28] and [29] ML must be embedded into dedicated platforms in order to reduce the power envelope constraint in wearable devices to the range of mW. In Chapter 4, we achieved a tactile object recognition implementation, consuming 300 mW, and less than 1 mJ energy.

Therefore, the key challenge is to improve the power consumption while preserving the real time constraints for longer battery life.

1.2.2 Energy Efficient Techniques

Saving energy can be applied on different layers of an architecture, i.e. starting from the algorithmic level by reducing the complexity, down to the hardware level. Therefore combining different optimization techniques, in addition to the choice of the correct hardware, will be useful to implement an energy efficient embedded AI algorithm .

Table 1.1 Energy Efficient embedded ML/DL algorithms on different hardware platforms [30]

Energy Efficient Technique	Design Approach	ML and DL algorithms	Hardware Platforms	Performance	Application
Parallelism	Intra-layer approach [23]	DCNN	Orlanco SoC	speed-up: 14.21x	
	Parallelization on 8 cores [31]		PULP-Mr.Wolf	Energy: 83.2 μJ Power: 10.4 mW	EEG
	Parallelization on 2 cores [32]	Tensorial SVM	PULP-Mr.Wolf	15x energy savings	E-skin
	Row stationary-Exploiting local data reuse [33]	CNN	Eyeriss Chip	1.4x - 2.5x energy savings	IoT Devices
	Pipeline through HLS directives [34, 35]	Linear SVM	FPGA	9.9x speed-up	Melanoma Detection
		Decision Tree			Character Recognition
	OpenCL (parallelism) [36]	KNN	FPGA	3x energy efficiency	KDD-CUP2004 Quantum physics set
Approximation	Quantization table-based matrix multiplication [37]	CNN and RNN	DNPU		
	Fixed point (1-9b) [38]	ConvNet		100x-energy savings	Wearable Devices
	Lower complexity network pruning [39]	Neural Networks	ASIC	5% till 87% power savings	CIFAR as benchmark
	Approximate Multipliers [39]				
	Approximate Memory for On-chip [39]				
	Inexact logic minimization approach [39]	Neural Networks	TSMC 65nm	43.9% till 62.5% energy savings	
	Improving external memory bottlenecks at the architecture level [40]	ConvNet	ASIC 28 nm silicone	2-9 TerraOps/W/s	Real-time embedded scene labeling
Network sparcity	Skipping Sparse operations [41]	ConvNet	Envision Platform	10 TOPS/W-efficiency	Wearable devices
	Energy aware pruning [42]	CNN	ASIC	70% power reduction	Wearable Devices and Smartphones
	Width and Resolution reduction [43]	MobileNet	ASIC	88% mult-add reduction 1% accuracy degradation	Image processing
	Removing zero operand multiplication [44]	DCNN	ASIC	1.24×-1.55× performance improvement	

Different energy efficient techniques were employed in the literature. In Table 1.1 a summary of these techniques and their usage at different levels, for embedded ML and DL implementation on different hardware platforms is presented. The main techniques are: Parallelism and data reuse, Approximation, and Network Sparsity. Finding the optimal combination or usage of these techniques, is a challenging task in order to reduce the power/energy consumption while still achieving the target application's requirements. In our perspective, a top-down approach can be followed in order to make this choice, e.g. starting by choosing the right algorithm, then making the optimisation at the algorithmic level, search a low power platform for the implementation, and finally apply a suitable techniques, details can be found on our work [45]. Moreover, new emerging technologies are going beyond the traditional computing architectures to make a boost in the computation while taking care of the energy consumption, like using new materials and architectures e.g. Nano-wires-based 3D stacked architecture [46],or tensorial computing architectures as well as task-specific accelerators, which are common in different applications e.g. CNN accelerators [47].

1.3 Contribution

In this perspective, the main contribution can be summarised as :

- Using transfer learning from visual image processing domain, into touch modality classification domain, by generating synthetic images from time series data coming from (4×4) tactile array, published in [48].
- Optimisation and Embedded implementation of CNN based tactile data classification neural network model, on various hardware platforms, published in [49]. This work includes experimental study about energy, latency and power consumption, and proved practically the feasibility of an embedded low-latency, low-energy tactile data processing solution, based on CNN.
- Implementation of small-size(memory), low-latency, low-complexity tactile data classification model based on shared-weights recursive neural network models e.g. LSTM and GRU, after applying a low-loss filtering and compression on tactile data, this work is submitted to IEEE sensors, under revision. This work as well, has reduced more than 99% of FLOPs, and more than 98% of model size with respect to the adopted model at our Labs for tactile data processing, which will open the opportunities to adopt this solution on embedded hardware in the next steps, with less processing and memory requirements. A simple pipeline architecture is presented in this solution, to make a simple parallelism between sensory data processing and acquisition, which also contributes in reducing the latency.
- An overview perspective for embedded implementation optimization, and energy efficient techniques, published in [30].
- An overview of approximate computing methods [45].

A complete list of all publications can be found in Appendix A.

1.4 Thesis Outline

The thesis will be structured as follows: In Chapter 2, Transfer learning for tactile sensing is explored, where real dynamic tactile touch modalities acquired from a 4x4 tactile array, for a 10 seconds duration, are transformed into a single static image and classified by CNNs using transfer learning. The work shows a higher accuracy with respect to the state-of-the-art (SoA), but with more complex model compared to the SoA. The proposed solutions achieved a best accuracy of 76.9% using Inception-Resnet.

Chapter 3 shows the use of LSTM and GRU for the same problem in the previous chapter i.e. touch modality specification. In this work data are processed as time series data using these two RNNs, a data organization process is proposed as well, the result was a higher accuracy, lower memory, and lower FLOPs compared to previous SoA solutions. The higher accuracy achieved is 84.23%,and a reduction of FLOPs by 99.98% and the memory storage by 98.34% compared to the SoA.

Chapter 4 studies a single model based on AlexNet for tactile data classification, and then different variations of input size are done. This study shows how smaller optimisations are enough to reduce the model size and complexity while keeping a good accuracy, best selected model in terms of accuracy and number of FLOPs is then implemented on different embedded hardware platforms. An embedded implementation is achieved as well on different hardware platforms. This solution achieved 11 to 43.6% decrease in the number of trainable parameters, 10 to 45.8% decrease in the number of FLOPs, with a change of accuracy in the range of [-5, 1.28]%
And finally, Chapter 5 contains the conclusion and future work.

Chapter 2

Touch Modality Classification through Transfer Learning

In this chapter, we demonstrate a method to achieve touch modality classification using pre-trained convolutional neural networks (CNNs). The 3D tensorial tactile data generated by real human interactions on an electronic skin (E-Skin) are transformed into 2D images. Using a transfer learning approach formalized through a CNN, we address the challenging task of the recognition of the object that was touched by the E-Skin. The feasibility and efficiency of the proposed method are proven using a real tactile dataset outperforming classification results obtained with the same dataset in the literature.

2.1 Introduction

The development of E-Skin systems has been motivated by the possible applications in many domains such as robotics, prosthetics, and biomedical applications [1]. Processing tactile information is a crucial task to respond to the application requirements (prosthetics or robotics) or to mimic the human skin behaviour. The processing is employed to extract tactile information e.g. material recognition, shape perception, grasping feedback or touch modality classification [1]. For instance, in prosthetic applications, the tactile information

can be processed within the prosthesis itself, then structured information are conveyed to the brain in order to close the loop and achieve a tactile feedback [50]. An important task to be addressed when developing the E-Skin system is to carefully select the appropriate method that provides efficient results responding to the application demands.

Machine Learning (ML) techniques based on large-scale neural networks, kernel concepts, or ensemble models have emerged recently as very promising methods assisting in predicting (recognizing or classifying) data or events using previously gained training experience. In tactile sensing applications, where the problem cannot be easily modeled, ML is a promising way to classify tactile data. In particular, deep neural networks (DNNs) have been applied recently to many domains such as image [26] and speech recognition [51], and medical image analysis [52]. DNNs have successfully produced results comparable to or overcoming previous techniques and, in some cases human tests [53]. Experiments mainly demonstrated that a DNN trained on a large number of images can be used to extract features from other images which are not part of the training dataset [54]. This transfer learning (or domain adaptation) process is accomplished through DNNs, by reusing a pre-trained model, previously developed for a task, as the starting point for a different task, and retraining a subset of its upper layers [54]. It provides a solution for cross-domain learning problems giving the possibility to benefit from data and models trained on a source domain, in order to extract features and make predictions on a target domain (Figure 2.1). In this perspective, this chapter takes benefit from the transfer learning capabilities of DNNs to propose a new approach to touch modality classification. E-Skin data resulting from: Brushing, sliding and rolling are reformulated in terms of a case-specific image format, and a DCNN is introduced and trained for transfer learning purposes, by exploiting both a small training set in the target tactile domain and a large pre-existing training set in a source domain of natural RGB images. The rationale behind this approach is: 1)On one hand, to benefit from the highly promising predicting power of deep net models. 2) On the other hand, to overcome the usual bottleneck

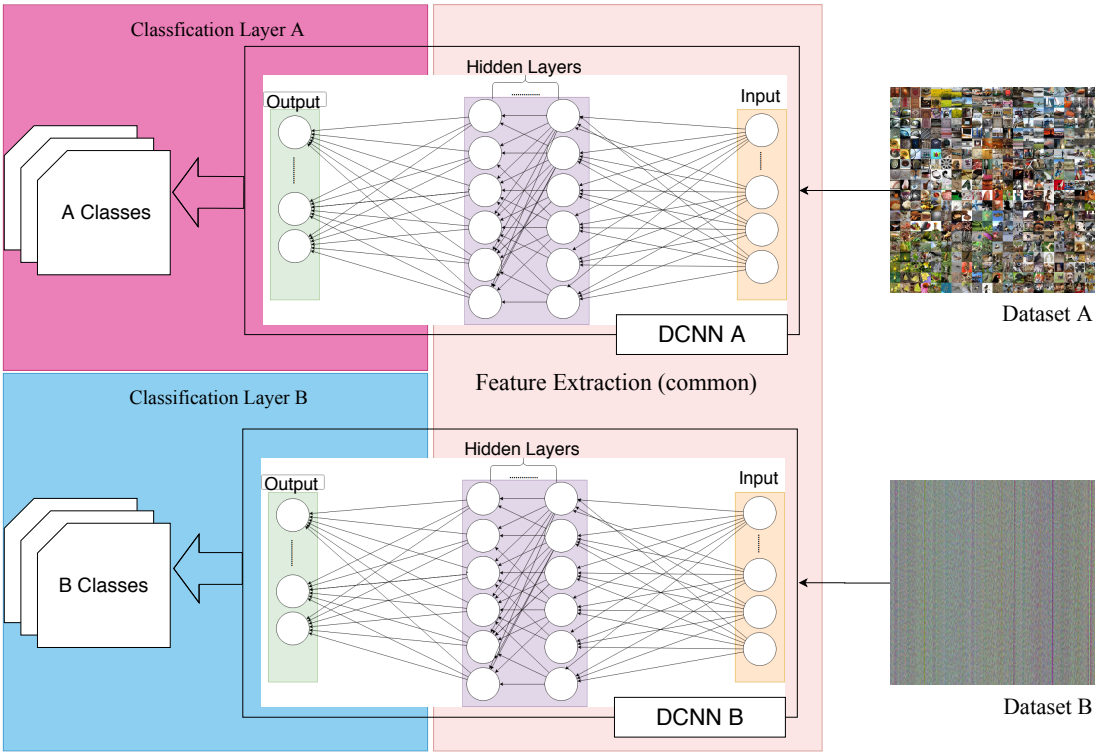


Fig. 2.1 Transfer Learning flow from domain A to domain B

in the use of deep learning in specific applications, i.e., the need for a large (most often huge) training set.

The rest of this chapter will be organized as follows: Related work will be exposed In the next section, Section 2.3 explains the used dataset and the proposed solution, while Section 2.4 discusses the results, finally in Section 2.5 the conclusion is presented.

2.2 Related Work

One of the first CNNs has been introduced in 1983 by Fukushima et al. [55] to recognize handwritten Arabic numerals. Another example was the handwritten zip code recognition in 1989 by Lecun et al. [56]. Later on different models have been introduced with different numbers of layers, input sizes, and outputs. Indeed, while methodological ideas were in

those early works already, highly successful CNN implementations have been much more recent and have been made possible by the dramatically increased availability of parallel and high-performance computing architectures [57]. Impressive results have been obtained in multiple applications [57] and international competitions (e.g. [58]). The layers commonly used in different CNNs are: Convolution, Pooling and Fully Connected layers. Convolution and Pooling layers are generally used for feature extraction, and fully connected layers for the final phase i.e. classification [57].

Various machine learning methods have been applied for tactile sensing systems targeting object [59] and texture recognition [60], or to estimate the grasping force based on slip detection in industrial applications [61].

Few works in the literature have addressed touch modalities classification. Silvera et al. [62] used Electrical Impedance Tomography (EIT)-based artificial sensitive skin, authors have used LogitBoost classifier to classify eight touch modalities i.e. 'tap,' 'pat,' 'push,' 'stroke,' 'scratch,' 'slap,' 'pull,' 'squeeze' and 'no touch'. Same modalities were applied to humans to compare artificial and human classification.

Deep CNNs (DCNNs) have also been used for tactile data processing. Gandarias et al. used a 25×80 tactile sensor array attached to a robotic arm to get RGB pressure images [63], they used a CNN to classify eight objects: finger, hand, arm, pen, scissors, pliers, sticky tape, and Allen key. Two approaches have been tested: the first with Speeded-Up Robust Features (SURF) descriptor, while the second one employs a pre-trained DCNN for feature extraction and a Support Vector Machine (SVM) for classification. The work presented in [64] deals with active clothing material perception based on an automatic robotic system for grasping the clothes. Authors used images coming from a large pressure sensor 18.6mm×14.0mm which produces RGB pressure images of 640×480 pixels. Using a pre-trained DCNN for feature extraction, the goal was to classify into 11 labels, those labels may have binary values (e.g soft or hard), or multiples values (e.g. discriminating among 20 textile types). In [63] and [64] the original data coming from sensors using a robotic setup

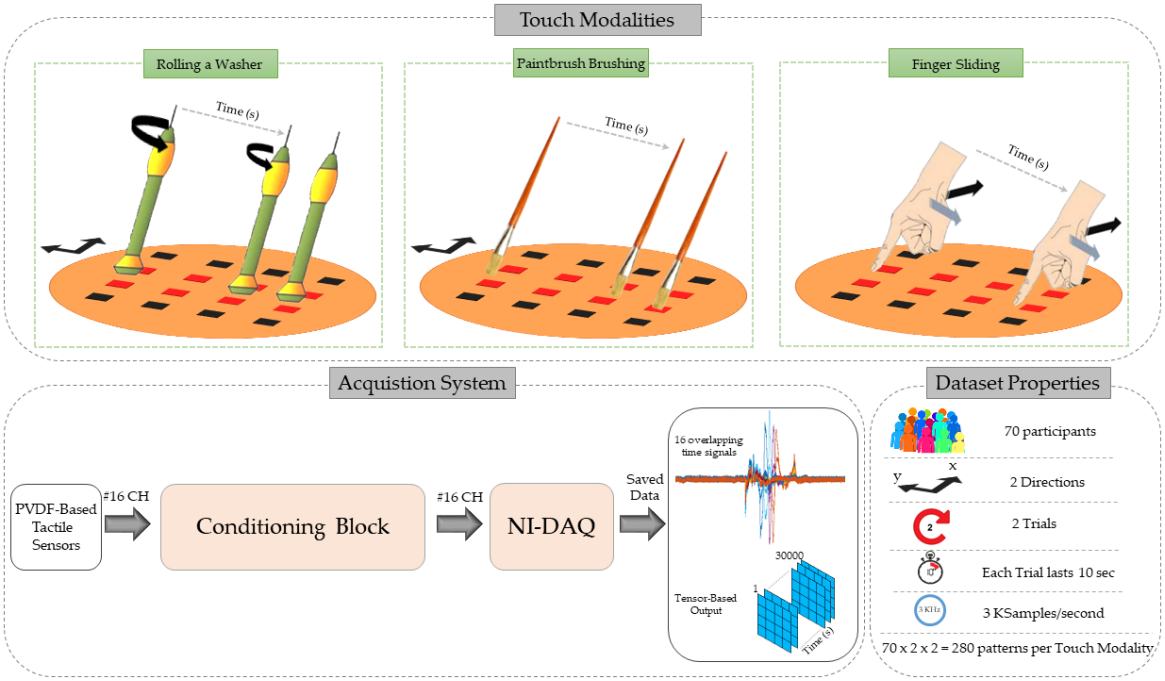


Fig. 2.2 Scheme of touch modalities, Tactile acquisition system and Dataset Properties.

are visually identified, because they describe the 3D representation of the objects. Kaboli et al. [65] worked on the same eight touch modalities presented in [62] for a multi-modal artificial skin composed of 32 cells attached to a robot: 16 cells on the front and 16 on the back, each cell has one local processor, one three axis accelerometer, one proximity sensor, three normal-face sensor, and a temperature sensor. The paper introduced new feature descriptors: Activity, mobility, complexity, linear correlation coefficient and non-linear correlation coefficient. Touch modalities were applied by humans to the robot's skin. SVM was used for classification, after finding optimal label parameters by 5 fold cross validation. Finally, Gastaldo et al. 2014 [66], used a 4×4 piezoelectric tactile sensor array; voltage levels generated by those sensors were collected over a definite time period, to distinguish between three touch modalities: Brushing a paintbrush, Rolling a washer, and Sliding a finger. Two algorithms were used: Tensor-SVM and tensor regularized least squares (RLS).

2.3 Approach and Methods

The experiments presented in this chapter have employed transfer learning to classify the same three touch modalities presented in [66]. The same dataset collected by Gastaldo et al. [66] have been used. Data were collected from 70 volunteers: each one tried three touch modalities: Brushing a paintbrush, rolling a washer, and sliding a finger. The same touch modality was repeated both horizontally and vertically by each volunteer twice, as illustrated in Figure 2.2 The total number of collected touch modalities is 840 (70 participants × 3 modalities × 2 trials × 2 directions). Tactile sensors are implemented by an array of 4×4 piezoelectric sensors, raw sensory data were recorded for 10 seconds for each trial, at a frequency of 3 kHz, which means that each trial output is a set of 4×4×30,000 samples (array dimension × time). Figure 2.3 shows the real sensory array's printed-circuit-board used in the experiment.

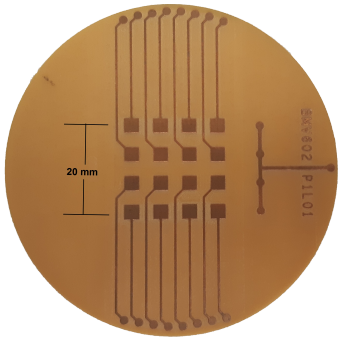


Fig. 2.3 Printed Circuit Board for the real sensory array used in the experiment.

In [66], only the second trial of each modality was taken into account, while the first try was not included in either training or testing. The rationale was that most volunteers were more comfortable in their interactions with the instrument on their second than on their first trials. Hence, the second-trial data were overall more regular and less noisy. Moreover, five participants were excluded since considered extremely noisy [66]. In this work all samples in the dataset have been included, i.e. no trials have been excluded. The dataset contains 840

tensors, i.e. 840 3D arrays of size 4×4×30,000 distributed evenly among: Brushing, rolling and sliding. Figure 2.4 shows a plot of a raw data for a rolling sample.

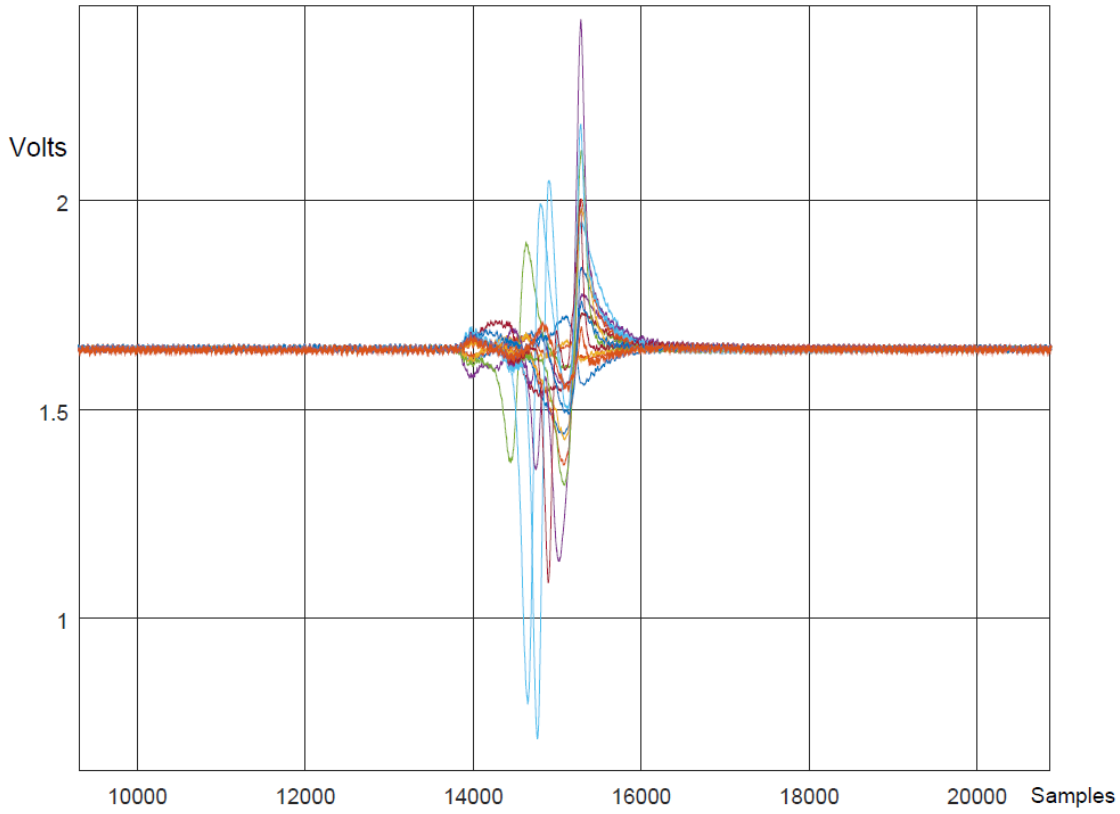


Fig. 2.4 Plot of raw data, where each channel (sensor), is represented by a color, here the touch starts around the sample number 14000, acquired at frequency of 3 kHz.

On one hand, the small width and height of the frames captured at once (4×4) prevents to successfully apply computer vision algorithms directly. Furthermore, in this application time is an important dimension, since a single 4×4 frame taken at a single time is generally insufficient to identify the modality, whereas the time series of many such frames can be discriminative enough.

On the other hand, CNNs are most often used to classify larger image size. Moreover, the pre-training of a CNN architecture with vast image databases is especially available in the case of natural RGB images. Therefore, in the proposed approach, a case-specific procedure has been defined to transform raw data coming from the tactile sensory array, i.e. 4×4×30,000

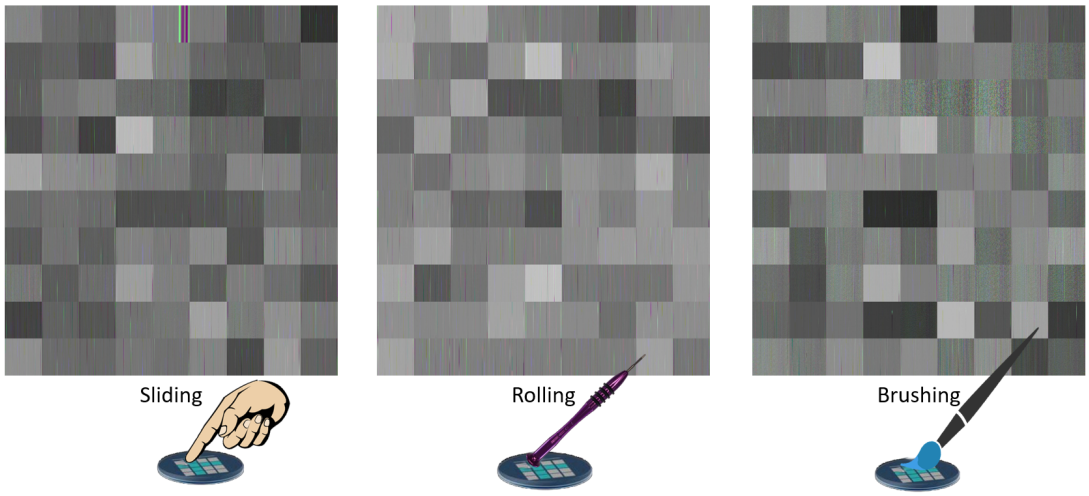


Fig. 2.5 Tactile images embedding, 90 samples for each class are embedded in a single image for visualisation only, they nearly impossible to be recognized by human eyes.

samples for each modality, were transformed into 400×400 RGB images, Figure 2.5 shows an embedding of the obtained images. In this way, larger images which have closer size to those used for CNNs pre-training without large scaling ratios (up or down), were obtained. The obtained images were fed to different CNNs trained on ImageNet [67] - a dataset containing millions of visually identified labeled images categorized into more than one thousand classes - in order to benefit from feature extraction layers. Only the last layer (i.e. essentially the classifier) is retrained using the aforementioned tactile training dataset.

Figure 2.1 illustrates this transfer learning concept from domain A to domain B, where the same feature extraction layers trained for domain A are used in the case of domain B, and then only the classification layer is re-trained with domain B data. The re-trained CNN is then used in the proposed solution; a general block diagram is shown in Figure 2.6.

2.4 Experiments and Results

The experiments for the proposed solution were made under the following setup:

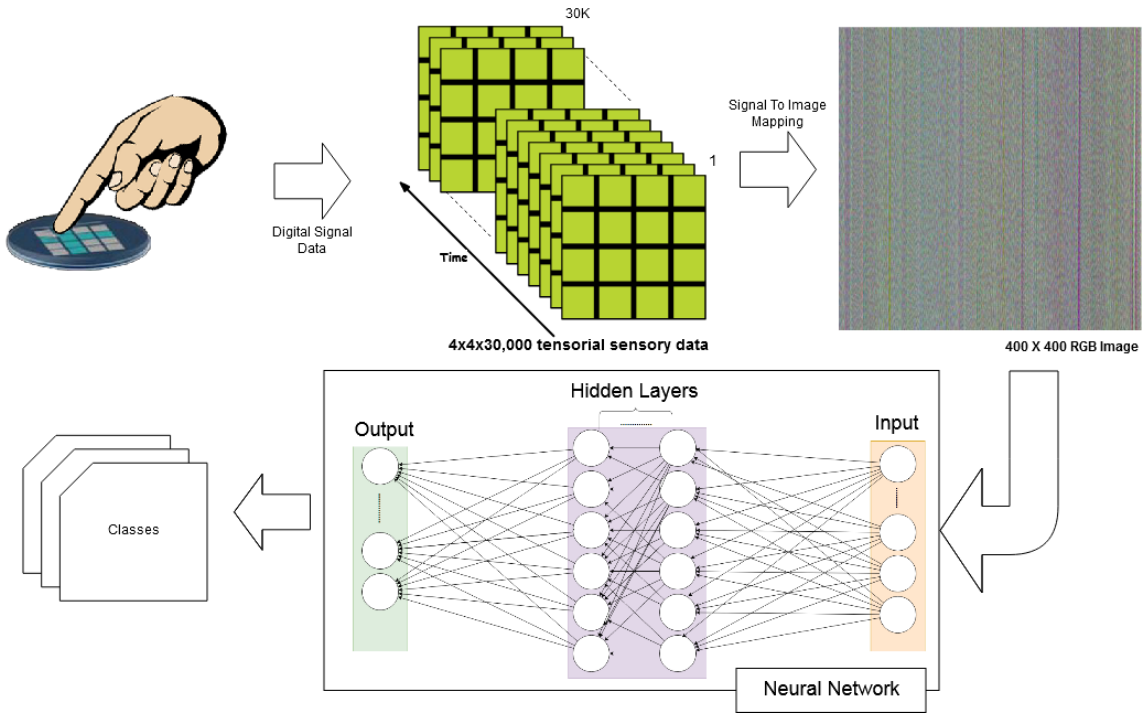


Fig. 2.6 Block diagram of the proposed solution

1. Hardware: Training and testing were made on Jetson TX2 development kit donated by Nvidia Corporation [68].

2. Software: Tensorflow 1.8.0 under Ubuntu 16.04 LTS.

3. Dataset: 840 images distributed evenly among the three classes: Brushing, sliding and scrolling.

4. Neural networks: ResNet v2-50, ResNet v2-152 [69, 70], MobileNet v2-035-2 [43], Inception v3 [71], Inception ResNet [72].

All experiments have been made using 80% of the dataset for training, 10% for validation (i.e for optimizing hyper-parameters), and 10% for testing. The training/ validation/ testing split was randomly selected. According to Figure 2.7, Inception ResNet (76.9%) and Inception v3 (75.8%) outperformed the average accuracy results in [66] for both Tensor-SVM (71%) and Tensor-RLS (73.7%). The latter results have been obtained by averaging the

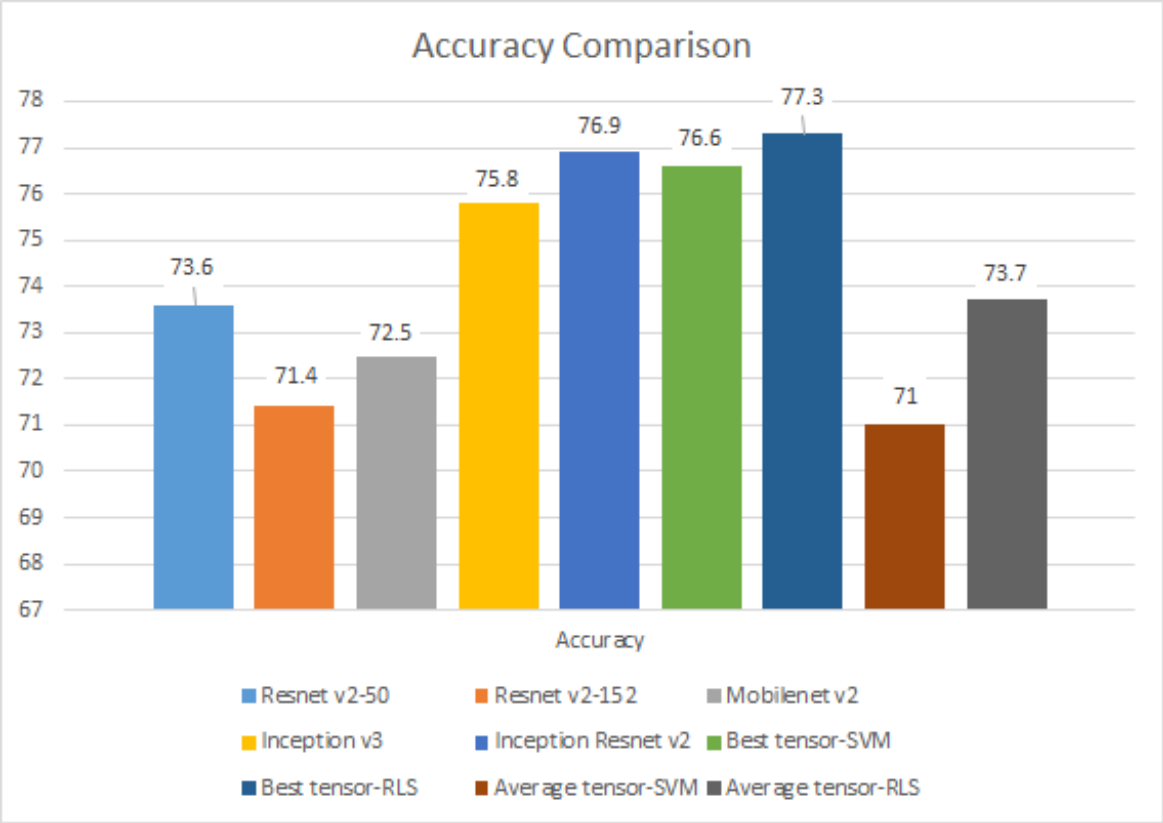


Fig. 2.7 Proposed solution vs Average tensor RLS/SVM accuracy

results of 5 different variations of tensor-SVM and tensor-RLS. Inception-ResNet outperformed also the best accuracy of tensor-SVM (76.6%), while tensor-RLS (77.3%), performed better than our best solution by 0.4% which is theoretically less than one sample in the experimental dataset. In [66], an effective model selection algorithm was also introduced to boost generalization performance, and only the second try of each modality was taken into account.

Compared to the work in [66], the proposed inference model is not affected by the number of training samples, i.e. the model size will remain intact in case more samples are provided for training. On the contrary, in the previous solution [66], three distinct models were trained in order to have three-class classification, and for N classes this number will grow at least to N models.

Moreover, feature selection was not introduced in the proposed method, instead tactile sen-

sory data were transformed into images, in order to benefit from automatic feature extraction achieved by DCNN in the proposed approach.

2.5 Conclusion

This chapter has proposed a DCNN approach for tactile sensory data classification based on transfer learning. The achieved results suggest the potential of current deep learning approaches to solve the challenging task of discriminating tactile touch modalities. Indeed, those results are very close to those obtained in human recognition experiments, which showed (e.g. in [65]) 71% human accuracy to discriminate touch modalities. Re-training a CNN designed for natural images, is proven in this work to be an efficient transfer learning solution to deal with non-image tensorial data collected by touch sensors over time. In the proposed solution classical tactile data classification [66] was outperformed by CNN, by transforming tensorial sensory data into images, then using transfer learning and CNN to achieve automatic feature extraction and then classification.

With the high progress and performance of CNN accelerators, which reached in some cases 4.4 TOPs/W [73], embedding the proposed solution into real prosthetic applications, opens new opportunities.

After proving the concept of the usage of deep learning in such problem, the complexity is still high, and the time latency as well (75ms), the dynamic energy consumption was 55.5mJ. In order to reduce the complexity of the proposed solution, another solution is proposed in the next chapter (Chapter 3), which relies on Recursive Neural Networks, and solve both complexity, latency and memory consumption

Chapter 3

Touch Modality Classification through Recursive Neural Networks

Recurrent Neural Networks (RNNs) are mainly designed to deal with sequence prediction problems and they show their effectiveness in processing data originally represented as time series. This chapter investigates the time series characteristics of RNNs to classify touch modalities represented as spatio-temporal 3D tensor data. In this chapter, different approaches are followed in order to propose efficient RNN models aimed at tactile data classification.The main idea is to capture long-term dependence from data that can be used to deal with long sequences represented by employing Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) architectures. Moreover, a case specific approach to dataset organization of the 3D tensor data is presented. The proposed approaches achieve a classification accuracy higher than state of art solution providing more effective performance in terms of hardware complexity by reducing the FLOPs by 99.989%. Results demonstrate that the proposed computing architecture is scalable showing acceptable complexity when the system is scaled up in terms of input matrix size and number of classes to be recognized.

Keywords: Tactile sensing systems, recurrent neural network, deep learning, tactile data classification.

3.1 Introduction

The adoption of tactile sensing systems in real world application is still limited and challenging [74], [75]. One key aspect is the complexity of the processing algorithms (basically the number of Floating Point Operations - FLOPs) when the hardware implementation is targeted. This affects mainly the energy consumption and time latency.

In this work, a novel touch modality classification framework using Recurrent Neural Networks (RNNs) is proposed. RNNs are mainly designed to deal with sequence prediction problems. They have been very successful in processing natural language, i.e., working on sequences of texts and spoken language that are represented as time series [76, 77]. Data acquired from tactile sensors have 3-dimension tensor structure (similar to a video) where the first two dimensions are defined by the geometry of the sensor array while time defines the third dimension. Hence, given this representation of tactile data, in this chapter we adopt RNNs as they are effective in processing time-series-data.

The main contributions of this work are summarized as follows:

- We explore the potential of RNN models for touch modality classification. For this purpose, we specifically propose two methods that are based on two separate RNN architectures, namely Long Short Term Memory (LSTM) [78] and Gated Recurrent Unit (GRU) [79] networks to capture long-term dependence from tactile data.

- We propose a case-specific approach for dataset organization to address the peculiarities of tactile data within the aforementioned architectures. For both LSTM and GRU models, averaging with overlap is applied to the input tensor aiming to maintain data about previous time-step in the current time-step.

The proposed RNN framework for tactile data classification has been experimentally validated with a real dataset and compared to the state of the art achievements. The achieved results demonstrate that the proposed approach achieves a classification accuracy of 84.23% on a 3-class touch modality data set (explained in Section 3.4), which is higher than state of art solutions [66, 48]. The proposed solutions reduce the number of FLOPs by 99.989%

compared to the same problem in the state of the art [32]. The computing architecture is scalable, i.e. the complexity is still acceptable when the system is scaled up in terms of input matrix size and number of classes to be recognized.

The rest of the chapter is organized as follows. Section 3.2 reviews the related works in tactile data processing. Section 3.3 gives a brief discussion on the methodology followed in the proposed approach. Section 3.4 introduces the experimental setup with the details of the different proposed models. In Section 3.5, we report and analyze the experimental results with a discussion. Finally, we conclude the chapter in Section 3.6.

3.2 Related Work

Different works in the literature have addressed tactile data processing using machine learning and deep learning, including the use of LSTM networks. In [63] two approaches are used to classify eight objects, i.e., finger, hand, arm, pen, scissors, pliers, sticky tape, and Allen-key, using a 28×50 tactile sensory array attached to a robotic arm. For feature extraction, the first approach uses Speeded-Up Robust Features (SURF) descriptor [80], while the second uses a pre-trained AlexNet CNN [81]. Finally, a Support Vector Machine (SVM) [82] classifier is employed for both approaches. In [64], a CNN is used for active tactile clothing perception. Color RGB pressure maps generated from a large tactile sensor attached to a robotic arm grasping clothes, are used to classify different textile properties: thickness, smoothness, textile type, washing method, softness, stretchiness, durability, woolen, and wind-proof. Different CNN models are experimented, the best performing being the VGG-19 pre-trained on ImageNet [67]. Rouhafzay et al. [83] employ a combination of virtual tactile sensors (32×32) and visual guidance to distinguish eight classes of simulated objects. Two neural networks are used: a 3D ConvNet for the series of object images coming from tactile sensors and a 1D ConvNet for the series of the normal vectors to the object surface. Abderrahmane et al. [84] introduce a zero-shot object recognition framework, to identify previously unknown

objects based on haptic feedback using BioTac sensors [85]. Two CNNs are employed: one for visual data and another for tactile data. In [86], authors use a shallow CNN (only three convolutional layers inside) based on AlexNet to identify 22 objects using pressure maps, collected from a 28×50 tactile sensory array. While in [49], an optimised embedded implementation of the latter solution is achieved on various hardware platforms. Various works on tactile data processing can be found in [87], [88], and [89] as well. In [90], authors collect frames of pressure maps from squeezing an object in contact with a TekScan 6077 tactile sensor(1700 taxels). 3D CNN are compared to 2D CNN in order to classify three different datasets achieving a higher accuracy when 3D CNN have been employed.

The aforementioned works generally have a high complexity in the learning and inference phases, especially when deep learning is used, which raises challenges for hardware implementation requirements [91]. Here arises the need of low-complexity, high-accuracy touch modality classification solutions suitable for the embedded hardware implementation, where resources are usually limited (e.g., power, memory). In this respect, LSTM is a promising candidate.

LSTM networks have recently attracted attention in tactile data processing, especially for the case of data presented in time-series, i.e., each sensor acquires time series of readings, defined by the data readout frequency. In [92], an LSTM is used to predict shape independent hardness of objects from data generated as video from a GelSight sensor with a grid of 960 × 720 pixels. Features from five video frames are extracted using a CNN and then used as an input for an LSTM network. In [93], authors use an LSTM for slipping prediction over six different material surfaces, using three different tactile sensors, attached to three fingers of a robotic arm. A 20-neuron single-layer LSTM is used in this work. In [94], authors use a CNN and a Graph Convolutional Network [95] for binary grasp stability detection, and an LSTM and a ConvLSTM for detecting slipping direction (translational and rotational). The number of LSTM units is not mentioned for LSTM, instead for ConvLSTM, five ConvLSTM layers were used, then pooling, and two fully connected layers, the input for ConvLSTM

is 11×12 RGB image. In [96], Dong et al. use high resolution tactile sensor (GelSlim) to control the insertion of objects in a box-packing scenario, they employ two distinct models based on CNN+LSTM, the first network for detecting the direction of the error, and the other for detecting the magnitude of the error, the CNN model used to extract features is a pre-trained AlexNet model; the LSTM contains 170 units.

While the aforementioned works address tactile data processing, few of them address touch modality classification. In [62] and [65], they classify nine touch modalities (tap, pat, push, stroke, scratch, slap, pull, squeeze, and no-touch) using LogitBoost [97] and SVM respectively. LogitBoost in [62] degrades in performance when trained on 40 participants (71%), while SVM achieves an accuracy range of [80.10 - 81.85%] in [65]. Gastaldo et al. [66] propose tensor-SVM and tensor-RLS to classify three touch modalities, while in our work [48] (Chapter2), we address the same problem using Deep Convolutional Neural Network (DCNN) and transfer learning. In the latter, we transform tensorial sensory data into synthetic RGB images, and use pre-trained CNN models on ImageNet [67] for feature extraction.

To the best of our knowledge, no works were done, yet, to explore the potential of LSTM for solving the touch modality classification problem.

3.3 Methodology

According to the state-of-art, different methods were used to address tactile data processing and touch modality classification. In this framework, the capability of deep learning architectures to extract meaningful data representations from high-dimensional spatio-temporal data, without the need for handcrafting features, conveys an especially promising potential. Here, we leverage on this potential to propose two novel methods for tactile data classification, based on recurrent architectures.

On one hand, the planar topology of a tactile array may generally prompt the use of CNN architectures. This strategy is expected to be promising especially if large-area arrays, made

of many individual sensors and acquiring tactile imagery with relatively high resolution, are used. On the other hand, highly effective CNN architectures may include a large set of parameters, which creates a challenge for real-time processing, power consumption, and training set size.

Touch modality data have relevant spatio-temporal characteristics: touch occurs in time (temporal aspect) and takes place on the surface of the tactile sensory array (spatial aspect). Using CNN in [48], necessitates to transform temporal characteristics into spatial characteristics, by generating a single synthetic image for each tensorial sample. Instead, RNN intrinsic capabilities to capture time-dependent behaviors makes them a promising approach for the analysis of such data. This is achieved by making a recursive input into the network, which comes from the output at previous time-step. Another important consequence of using RNNs is that the weights are shared across time, i.e., weights are defined for a single RNN block, and these weights will be shared upon execution over all time-steps. This means a reduction in number of stored trained parameters and of complexity as well.
Here, the RNN approach to the classification of touch modalities is explored and two RNN models (LSTM and GRU) are proposed. The description of each architecture will follow in the next subsections.

3.3.1 LSTM network

LSTM networks are RNNs capable of modeling long-range temporal dependencies [98]. RNNs are composed of a chain of units whose output is connected not only to the next layer but also fed back to the unit itself as an input, thus allowing the information to persist. LSTM behaves in a temporal manner that is appropriate for learning sequential models [99]. A clear example of LSTM usage is the prediction of the next word in a sentence, having observed the previous words [100]. Moreover, LSTM can act as a classifier for time series of data [101, 102]. A key characteristic of LSTM is its memory cell, which acts as an accumulator of

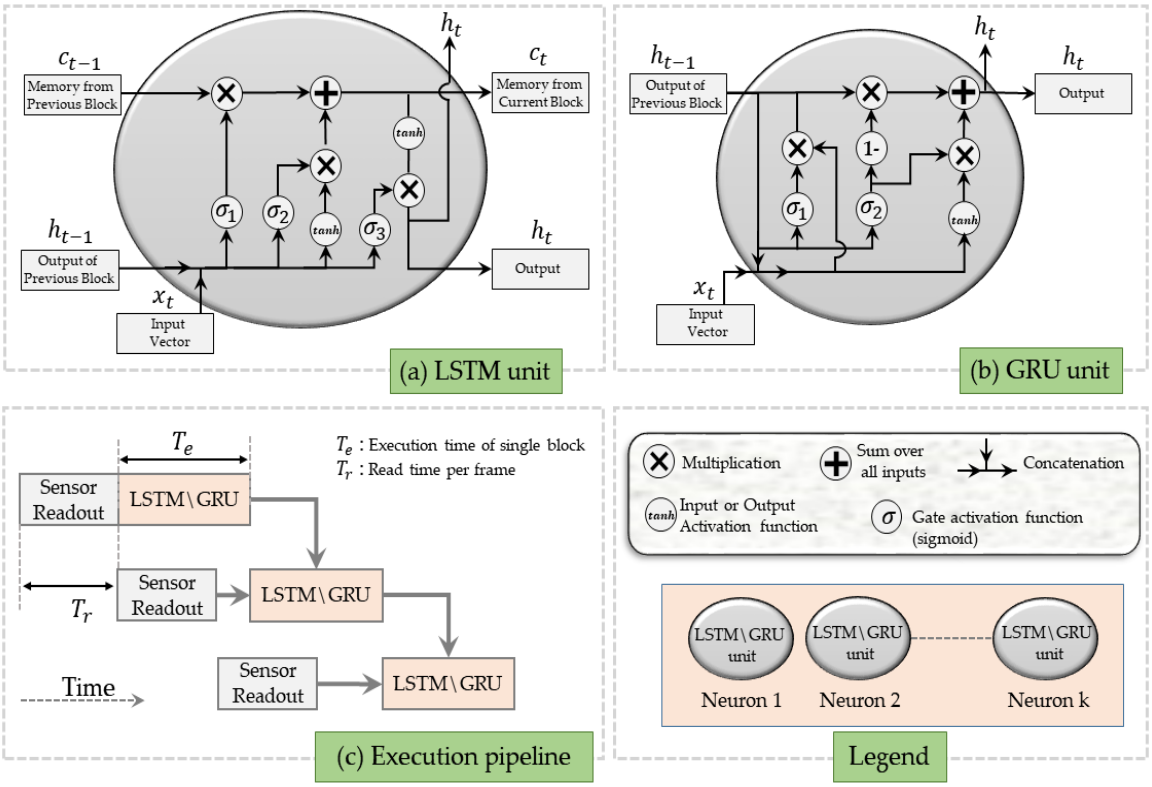


Fig. 3.1 (a) LSTM unit (b) GRU unit (c) LSTM\GRU Execution pipeline

the state information. Several self-parameterized controlling gates are used to access, write, and clear the cell (output, input, and forget gates).

Figure 3.1.a illustrates the architecture of the LSTM unit. It is composed of a cell, which is characterized by a state vector c_t and a hidden state vector h_t ($t \in \{1,2,\ldots,T\}$) indicates the time index and T is the number of time-steps. Each LSTM unit uses a forget gate, associated with a sigmoid activation function (σ_1), to decide which information it should forget from the previous state c_{t-1}. A new input x_t ($t \in \{1,2,\ldots,T\}$) is accumulated to the state of the memory cell c_t using a hyperbolic tangent activation function ($tanh$) and an input gate with sigmoidal activation function σ_2. At the end of the LSTM unit, an output gate using sigmoidal (σ_3) and tangential ($tanh$) activation functions are used to decide the outputs of the LSTM unit (c_t and h_t). Note that x_t is a vector with dimensionality equal to the number N of features, which represents the number of individual sensors in the array.

In order to get the number of parameters in an LSTM unit, the formula is the following :

$$p = 4 \times (H \times (D + H) + H) \tag{3.1}$$

where H is the number of hidden layers(neurons), D the dimension of the input vector (in our case 16). For each sigmoid function, the input is the concatenation of x_t and h_t, hence the total dimension of the input is $D + H$, this number is multiplied by number of neurons H, in addition we have H biases for each sigmoid function. so that the total number of weights of each sigmoid function is $(H \times (D + H) + H)$. Since an LSTM unit is composed of 4 sigmoid functions according to Figure 3.1, this number is multiplied by 4 as shown in equation 3.1. More details about LSTM networks can be found in [100].

3.3.2 GRU network

Similar to the LSTM unit, the GRU unit has gates that modulate the flow of information inside the unit. However, it does not use the memory cell state and uses the hidden state h_t to transfer information [79]. A typical GRU cell is composed of only two gates, the reset gate (whose role is similar to the forget gate of the LSTM) and the update gate (whose role loosely matches the input gate of the LSTM). Thus, a single GRU unit involves fewer operations and trainable parameters compared to a single LSTM unit. Figure 3.1.b shows the architecture of the GRU unit. In order to get the number of parameters in a GRU unit, the formula is the following :

$$p = 3 \times (H \times (D + H) + H) \tag{3.2}$$

The only different compared to equation 3.1,is the number 3 instead of 4, because the GRU block is composed of Three sigmoid functions instead of Four in the case of LSTM, as illustrated in Figure 3.1.

3.3.3 CNN-LSTM network

CNN-LSTM is a the series of a CNN and LSTM. The CNN-LSTM model is based on using pre-trained CNN layers, to produce a fixed-length vector representation of an input image to be used as a feature vector. This consists of cutting out the classification layer of the CNN, and just keeping all the layers before, whose final output is the feature vector. The feature vectors are then passed into an LSTM network, to implement the classification of a sequence of images. Figure 3.3 depicts the structure of the CNN-LSTM model.

3.3.4 ConvLSTM network

ConvLSTM networks [98] are used for capturing Spatio-temporal information in an image data sequence. These networks use convolutional layers inside their cells instead of fully connected layers used in standard LSTM networks. The main difference between LSTM and ConvLSTM is the type of operations performed in their units, but the logic keeps the same. That is, ConvLSTM networks still have a memory cell c_t that keeps a state at time t, and uses the same gates used in the LSTM unit to access, clear, and write the memory cell. However, a ConvLSTM operates with 3D tensors (e.g., RGB images) instead of 1D vectors (feature vectors) so it performs spatial convolutions with the data that go through it. Therefore,tensorial tactile images that presents a touch modality, can be used to train the ConvLSTM model and predict the type of such touch.

Based on the general architecture of the LSTM unit, three parameters are required to build an LSTM network: feature vector length, time-steps, and the number of neurons; the same applies to GRU as well. For these two RNN models, each input pattern is a feature vector representing a sampled time signal. In this perspective, raw data should be pre-processed in order to be suitable for RNN models. The pre-processing has three main objectives: 1) reduce the input data size to simplify the training, 2) normalise the data as a general consideration in training neural networks, and 3) make the dataset format compatible with the network's input format.

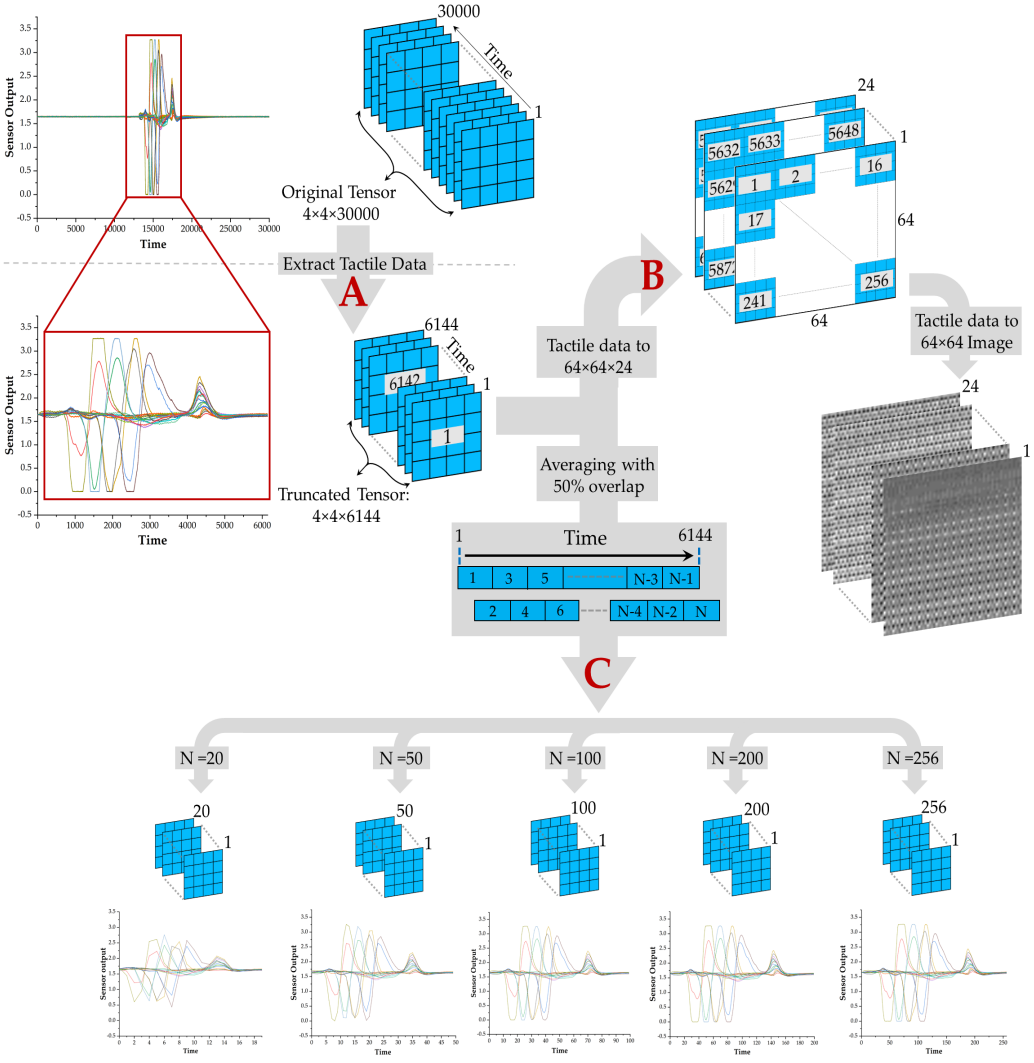


Fig. 3.2 Dataset Organisation

The feature vector length for both LSTM and GRU was chosen equal to the size of the tactile array, therefore each sensor in the tactile array is considered as a feature. The number of neurons was selected on trial basis, in order to have the fewest trainable parameters possible, while achieving an acceptable accuracy with respect to the state of the art. Detailed description of each model is presented in Section 3.4.B.

3.4 Experimental Setup

The dataset collected in [66] has been considered in this chapter. Seventy subjects were asked to perform predetermined touch modalities i.e. sliding the finger, brushing a paintbrush, and rolling a washer. Each participant touches the top surface of a 4×4 piezoelectric tactile sensory array in two moving directions twice. For every single touch, 10 seconds acquisition was done at 3 kSamples/second, the collected data were arranged into a 3-dimensional tensor: tactile sensory array size $\times$ number of acquired samples = $4\times4\times30000$. 280 patterns per touch modality in a total of 840 patterns are available. This dataset can be downloaded from: https://data.mendeley.com/datasets/dmcdp33ctt/2

3.4.1 Dataset Organisation

In order to use RNNs for this dataset, pre-processing was applied to the dataset. The first step is to generate a 3D tensor that contains only the useful touch information from the original raw data. In other words, in the first step, we selected the time period where touch is applied as shown in Figure 3.2.A. This was done by checking at which time instant T any of the sensor output value exceeds a predefined threshold. The resulting time T indicates the starting point of the useful touch data. To have a constant number of frames over all the patterns, we fixed the size of the data to 6144 samples per sensor, i.e., in the $[T, T+6143]$ range. The average activity time duration for all users is around 2 seconds i.e., 6000 samples. We selected 6144 samples per sensor i.e., 6144 $\times$ 16 to make the tensor size a multiple of

64×64, which make it suitable for comparison with CNN based networks, mentioned in section 3.4.B.

Algorithm 1: Generate Dataset (C)

```
Input: Dataset (A), Output: Dataset (C)
a = single pattern from Dataset (A);
K = number of frames in a;
C = output pattern;
F = number of sensors;
N = number of output frames;
slot ← K/N;
for i ← 1 to F do
    sen ← a(1 : end, i); // sen: single sensor output
    u ← 1;
    for j ← 1 to N do
        if u < N then
            C(j,i) ← Average(sen(u : slot + u));
        if u = N then
            C(j,i) ← Average(sen(u : end));
        u ← u + slot/2;
```

The result from the first step is a 3D tensor of size 4×4×6144 each, having the same original frequency of 3 kSamples/second, and starting at time instant T until reaching 6144 frames. Sixty patterns out of 840 were excluded since no sensor readings exceeded the activity threshold in these patterns. The refined dataset referred later on as Dataset (A), is composed of D×(4×4×6144) tensors per touch modality(three touch modalities), where D is the number of patterns ($D = 260$). From Dataset (A) are then derived two different datasets that fit the used models.

CNNs use convolutional layers to transform images into feature vectors, following that requirement, each pattern in Dataset (A) of size 4×4×6144 has been transformed into a time series of larger images. Each pattern is presented by 64×64×24 samples (image size × time-steps). The resulted dataset is called Dataset (B). It is important to note that the time sequence order was also maintained within each 64×64 image (im_{64}), i.e., each im_{64} consists of 256 images of 4×4 pixels as shown in Figure 3.2.B.

The third dataset called Dataset (C) is composed of five different sub-sampling of Dataset (A), as shown in Figure 3.2.C. An averaging with 50% overlap was used to down-sample the tensor from $4 \times 4 \times 6144$ to $4 \times 4 \times N$, where N represents the number of the input time-steps for the LSTM and GRU networks. N varies in the following set of values 20, 50, 100, 200, and 256. The overlapping helps maintaining data about previous time-step in the current time-step, the increment of (slot/2) in the sub-sampling algorithm clarify this overlapping. The whole process is described in Algorithm 1.

The three datasets are shown in Figure 3.2, where one pattern was used as an example to illustrate the difference in the presentation of the original dataset. The pre-processing code is presented in Appendix C.1.

3.4.2 Implementation

As mentioned in Section 3.3, two models have been implemented. For each model a dense layer is added at the end for the classification. For a further comparison, two CNN-based models are tested.

LSTM network

the LSTM network composed of one LSTM layer (10 neurons) and a flat input layer of length 16, was trained on Dataset (C). As for the time-steps, the LSTM network was trained for each of the time-step configurations i.e., $N \in \{20, 50, 100, 200, 256\}$.

GRU network

The GRU network is composed of a single GRU layer, applied within two alternatives: 10 neurons and 12 neurons per GRU layer. This GRU network has as well a flat input layer of length 16. As for the time-steps, the training process was done only on ($N = 20$) configuration of Dataset (C), based on the best achieved results with LSTM.

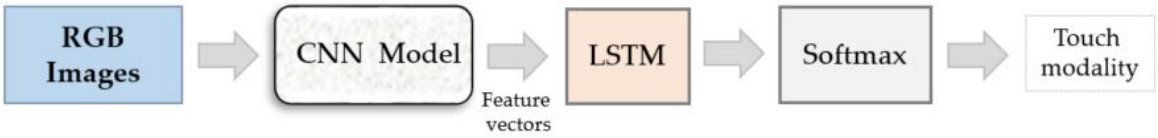


Fig. 3.3 CNN-LSTM Structure.

CNN-LSTM network

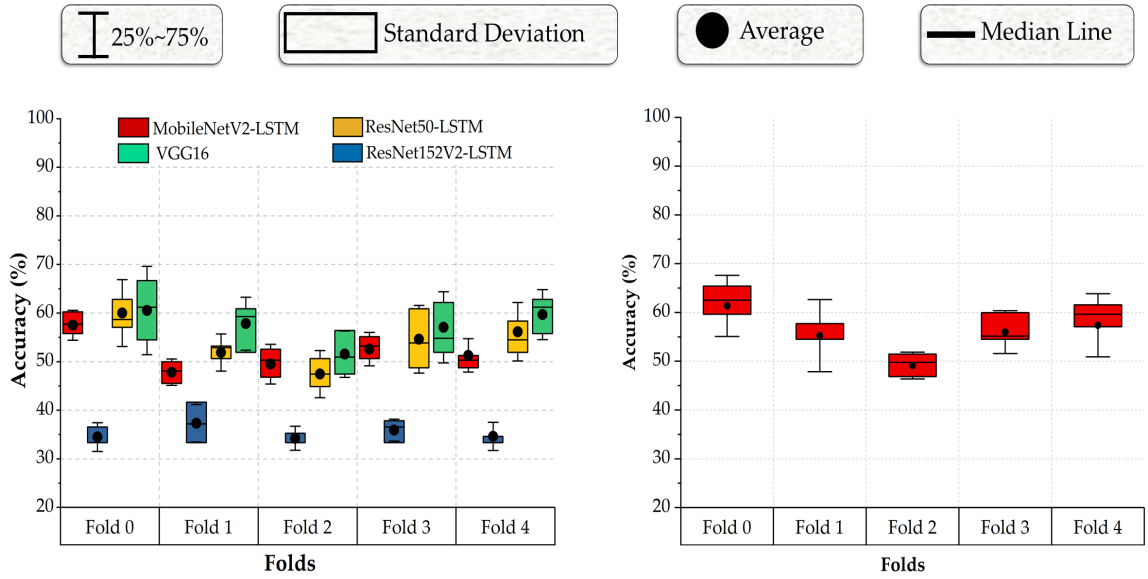
Four different pre-trained CNN models (ResNet50 [103], ResNet150V2 [70], MobileNetV2 [43] and VGG16 [104]) were considered for comparison purposes to play the role of the CNN model shown in Figure 3.3. The four CNN models were trained using the ImageNet dataset [67]. These models were selected based on the results of our previous work [48], which showed that the four models were effective in extracting features from the considered dataset. The models were used to transform each image in Dataset (B) into a fixed-length feature vector. Therefore, each CNN model transform Dataset (B) into a 4-Dimension tensor of shape $D \times 24 \times 1 \times K$, where D is the number of patterns, K is the size of the output feature vector in each model, and 24 is the time-steps in each pattern. Thus, the input of the LSTM block has K features, and 24 time-steps.

ConvLSTM network

The second CNN-based model used for comparison is composed of single ConvLSTM layer (32 filters of 3×3) followed by one fully-connected layer with 100 units and ReLU activation. Dataset (B) is used, the input of the network is a im_{64}, and the time-steps are 24.

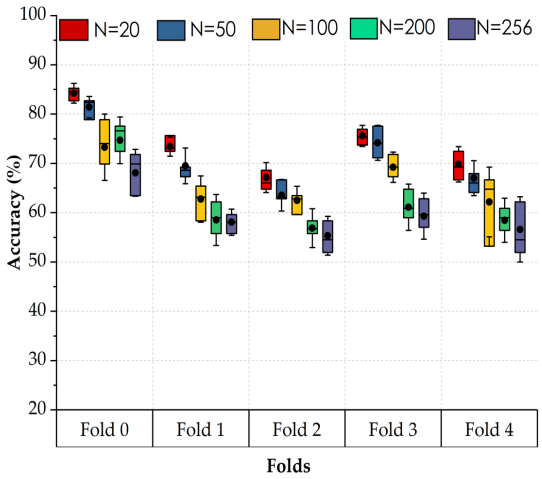
3.4.3 Training

The training and application of the proposed networks were done using Keras with Tensorflow back-end on an NVIDIA GPU. Dataset (C) was normalized before being used to train and test the LSTM and GRU models. Full code can be found in Appendix C.2.

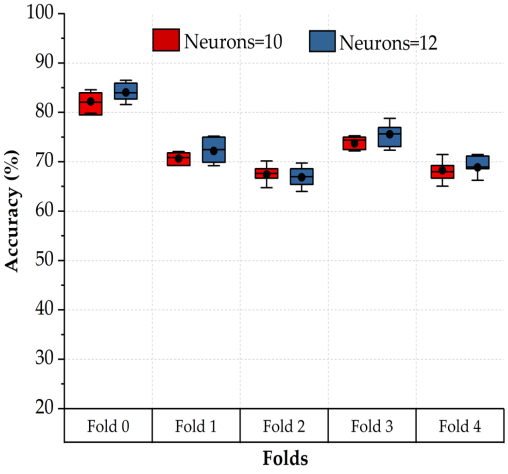


(a) CNN-LSTM

(b) ConvLSTM



(c) LSTM



(d) GRU

Fig. 3.4 (a) Average Accuracy achieved varying the input to the CNN-LSTM network for five different folds of Dataset B; (b) Average Accuracy of ConvLSTM network for five different folds of Dataset B; (c) Average Accuracy achieved varying the input to the LSTM network for five different folds of Dataset C; (d) Average Accuracy achieved varying the number of neurons in the GRU network for five different folds of $N = 20$ Dataset C.

For training / testing split, an 80 / 20 percentage was chosen. Five folds were generated, in a way that the intersection of testing samples is empty across all folds. The Adam optimiser [105] was used to train the networks, with categorical cross entropy as a loss function and Softmax activation for the output layer. Different runs were made also for each fold, including different batch sizes and epochs, in order to find the hyper-parameters that lead to better accuracy. Finally the choice was limited to $batch\_size = [48, 69]$ and $epochs = [48, 96]$, such that we have 4 combinations in total. For each combination, ten training runs with random initialization and random batch selection have been made i.e., the batch size is fixed, but choosing the samples for a batch is random. Therefore, for each model mentioned in 3.4.B, $4 \times 10 \times 5$ (combinations$\times$runs$\times$folds) training runs have been made. Finally the accuracy is obtained by averaging all runs across a fold for all the combinations of $(batch\_size, epochs)$. Results in the next section corresponds to the best $(batch\_size, epochs)$ combination, i.e., the combination that gave the highest accuracy, in our case it is $batch\_size = 48$ and $epochs = 96$. Figure. 3.4 shows the accuracy obtained on each fold, for the selected $(batch\_size, epochs)$ combination, using the four different models.

3.5 Results and Discussion

According to Figure 3.4.c, the use of LSTM with $N = 20$ time-steps referred later as LSTM20, i.e. using the Dataset (C) with sub-sampling into 20 samples for each pattern, shows a higher accuracy according to other sub-sampling alternatives, and with respect to other tested models. LSTM20 has achieved the highest accuracy: 84.23%, the smallest number of trainable parameters: 1113, and the smallest number of FLOPs: 2950 per LSTM block as shown in Table 3.1.

Regarding the GRU, GRU20 also proved a high accuracy with a smaller number of parameters and a comparable number of FLOPs with respect to LSTM20 as shown in Figure 3.4.d. The 10-neuron GRU achieved an accuracy of 81.92% with 843 trainable parameters

Table 3.1 Comparison of accuracy, number of parameters, and FLOPs

Model	Best Accuracy	Average Accuracy ± Stdev (%)	Model parameters	FLOPs
ResNet50-LSTM	67.56	60.82 ± 6.74	26M	107M
ResNet150V2-LSTM	37.58	34.48 ± 3.10	61M	246M
VGG16-LSTM	69.67	62.39 ± 7.28	15M	62M
MobileNetV2-LSTM	60.76	57.5 ± 3.26	4M	17M
ConvLSTM	65.56	59.3 ± 6.26	314M	1G
LSTM20 (10 neurons)	**84.23**	74.02 ± 6.56	1113	2950×20
LSTM50 (10 neurons)	79.84	70.72 ± 6.24	1113	2950×50
LSTM100 (10 neurons)	73.26	65.98 ± 5.01	1113	2950×100
LSTM200 (10 neurons)	74.67	61.92 ± 7.29	1113	2950×200
LSTM256 (10 neurons)	68.07	59.47 ± 5.03	1113	2950×256
GRU20 (10 neurons)	81.92	72.06 ± 6.60	**843**	**2228×20**
GRU20 (12 neurons)	**83.78**	73.07 ± 6.53	1083	2960×20
Tensor-SVM [66] [32]	76.6	71±5.6	67200	545M
Tensor-RLS [66]	77.3	73.7 ± 3.6	-	-
DCNN (InceptionResNetV2) [48]	76.9		54M	109M

and 2228 FLOPs. While a 12-neuron GRU, with 1083 trainable parameters, and 2960 FLOPs per single GRU block achieved 83.78% as shown in Figure 3.5 and Table 3.1. Both the GRU and LSTM models have achieved an accuracy higher than the best accuracy achieved by state-of-the-art approaches applied to the same dataset, whether in tensor-SVM (76.6%) and tensor-RLS (77.3%) [66], or using DCNN (76.9%) [48]. If we take the average accuracy, we can see that LSTM20 has achieved the higher average accuracy across all of them.

As per ConvLSTM and CNN-LSTM, these models did not converge well, and the obtained accuracies did not exceed the 70% as shown in Figure 3.4.a and Figure 3.4.c. One reason behind it, is the large number of features and therefore trainable parameters, compared to the small dataset size. Notwithstanding that, the previous results in DCNN [48] were higher, but that was done using transfer learning, i.e. all the used networks (except the classifier), were pre-trained on millions of images from ImageNet [67] and thousands of classes, then a classifier was trained on the subject dataset. Instead in CNN-LSTM we are using pre-trained CNNs to extract features from Dataset (B), the resulting feature vector may range from 2K (VGG16) to 8K (Resnet150v2) features. These features are fed into an LSTM

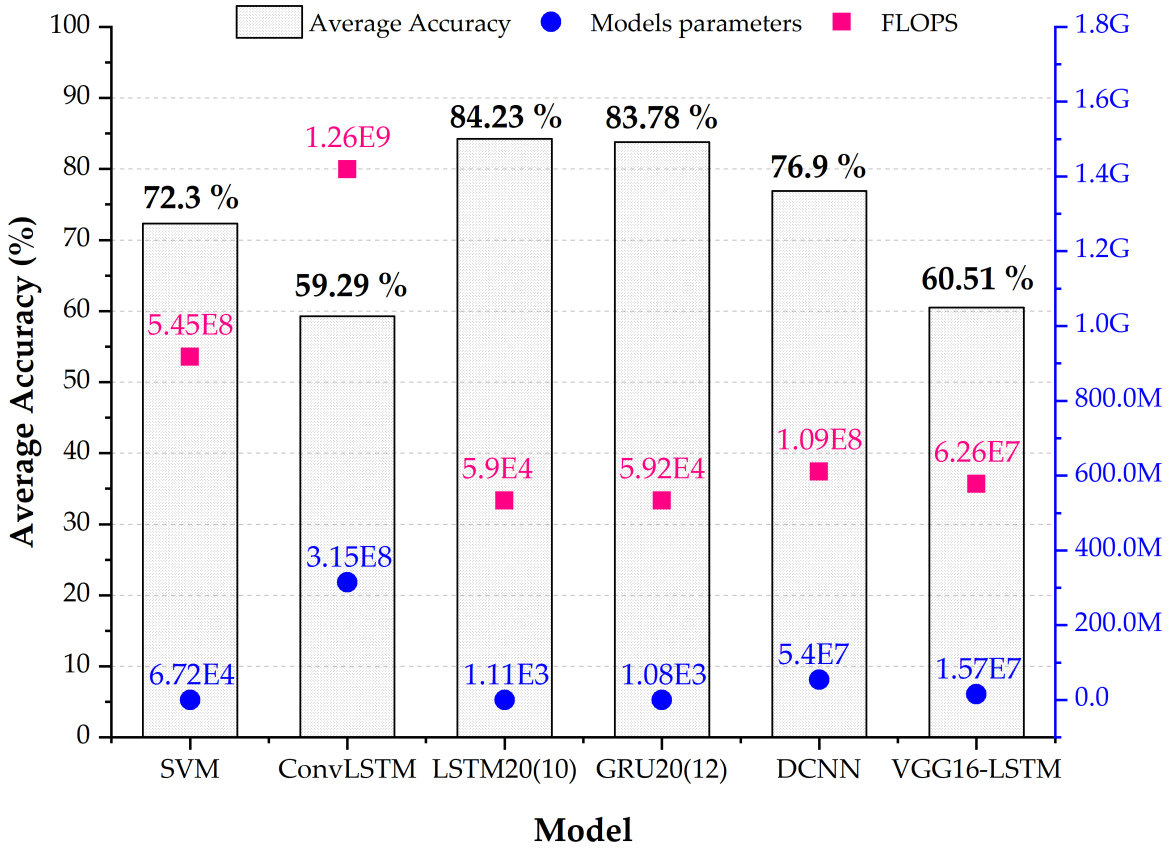


Fig. 3.5 Accuracy of the best performing networks. LSTM20(10) stands for LSTM network with 10 neurons and $N = 20$. GRU20(12) stands for GRU network with 12 neurons and $N = 20$.

network to train it from scratch, compared to 16 features in LSTM or GRU with Dataset (C), which induces higher number of trainable parameters for both CNN-LSTM and ConvLSTM compared to GRU and LSTM, as shown in Table 3.1. Raising the number of LSTM layers and number of neurons for ConvLSTM and CNN-LSTM lead to better results, but still not reaching a comparable accuracy.

LSTM achieved many benefits:

1) FLOPs and memory occupation i.e., number of trainable parameters are less, compared to the SOA.

2) As for the computation, data can be fed into an LSTM network, as soon as all features are ready for a single time-step, i.e., when data are ready at time t, they can be forwarded into

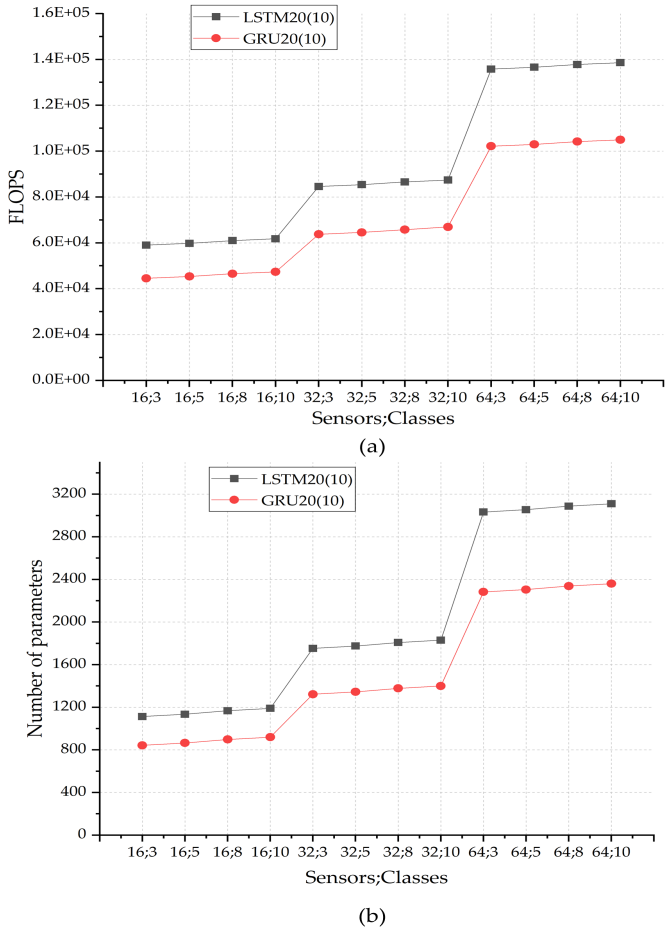


Fig. 3.6 (a) Number of sensors and classes versus number of FLOPS. (b) Number of sensors and classes versus Model parameters.

an LSTM block, without waiting for the data from all time frames. Also, data occuring at time $t+1$ can be collected in parallel with respect to the execution of LSTM block of data at time t, as illustrated in Figure 3.1.c. Unlike other solutions like CNN or tensor-SVM, all data should be assembled before bein processed.

3) Higher accuracy is obtained.

4) The model is highly scalable and independent on the size of the dataset, in terms of both FLOPs and number of trainable parameters, as illustrated in Figure 3.6. Unlike SVM, where the model size depends on the number of training samples and does not support multi-class labeling directly [106]. In addition, since LSTM used shared weights, e.g., when training an

LSTM of $N = 20$ time-steps, weights are shared across all time-steps, which makes a model trained on N time-steps data still usable for M time-steps data.

3.6 Conclusion

In this chapter, we have investigated the potential of RNNs for touch modality classification. Two different approaches based on LSTM and GRU architectures have been proposed to extract long-term dependence from tactile data. Proposed approaches have been validated on real tactile data acquired from 4×4 piezoelectric tactile arrays. Experimental results have shown that the achieved accuracy is of 84.23% and 83.78% for LSTM and GRU respectively compared with a value of 76.9% for the best achieved accuracy in literature [66, 48]. On the other hand, the proposed architectures reduce drastically the number of FLOPs considered as the main factor affecting the hardware complexity of the system. The number of FLOPs has been reduced of 99.989% compared to the same problem in the state of the art [32] which will have the impact on time latency, hardware resources, memory storage and energy consumption when the hardware implementation will be targeted. Another important aspect offered by the proposed approach is the scalability in the computing architecture. This means that the complexity of the system remains acceptable when the system is scaled up in terms of input matrix size and number of classes to be recognized which was a main drawback limiting similar state of art solutions [106]. To mention that the proposed solution achieved less than 5ms latency time on NVIDIA GPU, compared to 75ms for the same problem using DCNN in Chapter2. As a conclusion, the proposed work represents a good candidate to be embedded together with tactile sensing system for robotic or prosthetic applications [1, 2]. Such applications require near-sensor processing with critical constraints such as small hardware area, low energy budget due to the limited battery size, and the low latency needed to perform real time functions.

Chapter 4

Embedded CNN Implementation for Tactile Object Recognition

Embedding Machine Learning methods into the data decoding units may enable the extraction of complex information, making intelligent the tactile sensing systems. This chapter presents the efficient implementations of a Convolutional Neural Network model on different hardware platforms for tactile data decoding. Experimental results show comparable classification accuracy of 90.88%, overcoming similar state of art solutions. In terms of time inference, the proposed implementation achieves a time inference of 1.2 ms while consuming around 900 µJ. Such embedded implementation of intelligent tactile data decoding algorithms enables real-time tactile sensing systems in different application domains such as robotics, and prosthetic devices.

4.1 Introduction

Embedding intelligence near to the sensor location may enable tactile sensing systems to be incorporated in many application domains such as prosthetics, robotics, and internet of things. Decoding tactile information concerns different kind of tasks which could be categorized as simple or complex processing, depending on the algorithm's complexity. For the simple

processing, an example of the information retrieved is temperature, intensity of the contact force, contact location, direction and distribution. Concerning the complex processing, more intelligent tasks are expected such as patterns, textures, and roughness classification, or touch modalities discrimination. Employing the complex processing approach enables intelligence in tactile sensing systems. It is achieved by applying some complex data decoding algorithms able to extract the meaningful information from sensors e.g. Machine learning (ML), and Deep Learning (DL).

However, embedding machine learning algorithms on hardware platforms near to the sensors location is challenging due to the complexity such algorithms impose in terms of time latency and energy consumption. Our main goal is to achieve a tactile sensing system able to perform AI tasks. This system is intended to be portable/wearable in which the energy budget is limited. Moreover, for the target applications i.e. robotics and prosthetic, the lightweight is a critical constraint limiting the hardware and battery size.

In this perspective, this chapter presents the implementation of CNN algorithms on different hardware platforms. The main contribution of this chapter may be summarized as follow:

- It proposes an optimized CNN model, adopted and used from [86], based on reduced data which demonstrates to provide comparable results in terms of accuracy i.e. 90.88% with reduced hardware complexity.

- It presents efficient implementations of the CNN model on different hardware platforms for embedded tactile data processing. Proposed implementations achieve a time inference of 1.2 ms while consuming around 900 µJ. The work demonstrates its suitability for real-time embedded tactile sensing systems.

- It raises discussion about integrating intelligence into tactile sensing systems and how it enables tactile sensing systems in different application domains.

The remainder of this chapter is organized as follows: Section 4.2 reports the state of the art showing the recent embedded CNN implementations; In Section 4.3, we illustrate the experimental setup and methodology; In Section 4.4 the hardware implementation is explained; Results and discussion are presented in Section 4.5, followed by conclusions in Section 4.6.

4.2 Related Work

Different works have addressed the tactile data classification problem, using different methods including and not limited to machine learning, and deep learning [2, 107–110]. While most of work done was focusing on the methodology itself, few addressed the implementation on embedded platforms where should reside the real application. Other related works are mentioned in the previous chapters like [63], [64], [83], [84, 111], [87–89]. In [86], they used light CNN based (only 3 Convolutional layers inside) on AlexNet, to identify 22 objects using their pressure map, collected from a 28×50 tactile sensory array, we will use the same dataset for the embedded implementation.

While all these previous works were not implemented in an embedded environment, we can find few others targeting embedded implementation for tactile sensing applications. The need of embedded implementation arises from the need of having low power, small form factor electronics to process the tactile information, especially in prosthetic applications [50]. Osta et al. [32] demonstrated an energy efficient system for binary touch modality classification, based on SVM and implemented on custom hardware architecture, the energy per inference was 81mJ, and the inference time is 3.3 s. Ibrahim et al. [106] presented a real-time implementation on FPGA for touch modality classification, using SVM they achieved 350 ms inference time and 945 mJ inference energy for 3-class classification, and 970 ms/ 6.01 J for 5-class classification.

4.3 Experimental Setup and Methodology

4.3.1 Dataset

Targeting the classification of tactile data, the use of the dataset collected in [86] is considered. Tactile data have been collected by a high resolution (1400 pressure taxels) tactile array which has been attached to the 6 DOF robotic arm AUBO Our-i5 [86]. A set of piezo-resistive tactile sensors are distributed with density 27.6 taxels/cm^2 forming a matrix of 28 rows by 50 columns. The dataset is composed of pressure images that present the compliance of 22 objects with the tactile sensors. These images are divided into 22 classes labeled as Adhesive, Allen key, arm, ball, bottle, box, branch, cable, cable pipe, caliper, can, finger, hand, highlighter pen, key, pen, pliers, rock, rubber, scissors, sticky tape, and tube. Figure 4.1 shows an example of tactile images of three objects used for the training of the CNN model. Each taxel in the tactile array presents a pixel in the pressure image, thus each pressure image is of size $28\times50\times3$. Therefore the color of the pixel presents the pressure applied at the corresponding taxel. Where the minimum pressure is presented by Black color and the maximum pressure is presented by red color. Pressure images are then transformed into gray-scale images (image size=$28\times50\times1$) forming the tactile dataset.

4.3.2 Tested Model

Due to computational and memory limitations in the embedded application, a light CNN model is required to perform classification tasks with high accuracy and less number of parameters. In this work, we choose to use one of the models implemented in [86] as a base model to classify the objects in the aforementioned dataset. Among all the implemented networks we chose to use the custom network TactNet4 because it is the best network that fits the embedded application (less number of parameters with high accuracy [86]). The model is based on AlexNet which is usually used in computer vision for object classification [112]. The network is composed of 3 Convolutional layers ([Conv1, Conv2, and Conv3]) with filters

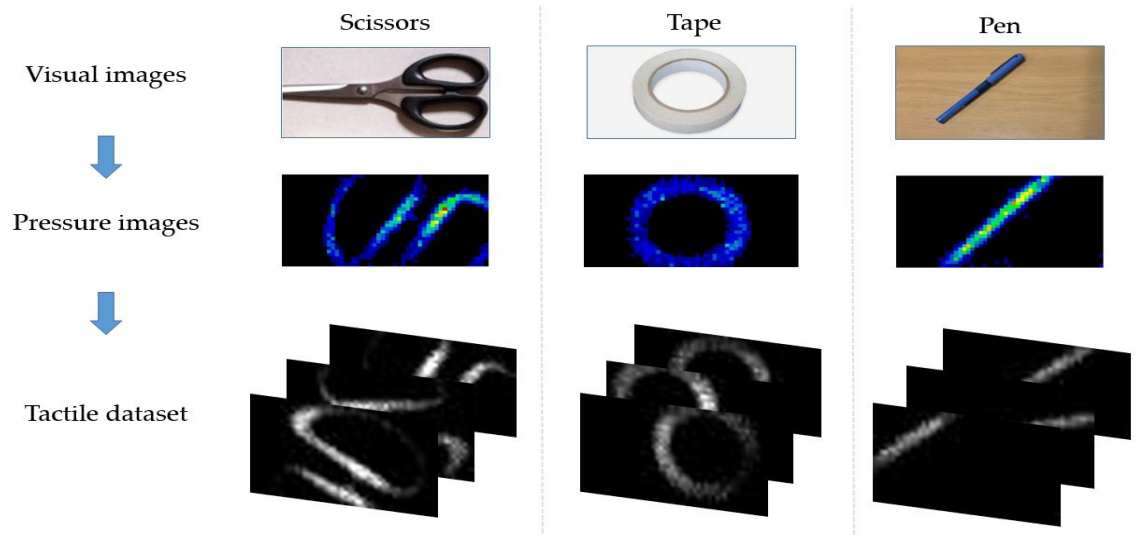


Fig. 4.1 Examples of visual (top) vs pressure (Middle) vs Tactile images (bottom) for common objects.

sizes ([5×5], 8), ([3×3], 16), and ([3×3], 32) respectively. Each Convolutional layer is followed by Batch Normalization (BaN), Activation (ReLU), and Max Pooling (Maxpool) layer respectively, where all pooling layers use 2×2 max-pooling with a stride of two. A fully-connected layers (FC = [fc4]) with 22 neurons followed by a Softmax layer are used to classify the input tactile data and give the likelihood of belonging to each class (object). The input shape of the model is configured to the size of the collected tactile data. Figure 4.2 shows the detailed structure of the used network. The network has been implemented in Matlab R2019b using the Neural Network Toolbox. A total of 1100 tactile images have been used to train the model. The learning process has been implemented on Matlab by dividing the tactile data into three sets: training, validation, and test set.

When having an adequate dataset, the validation set is expected to be a good statistical representation of the entire data set. If not, the results of the training procedure highly depend on how the dataset was divided.

To avoid this, In this work, we have used cross-validation method. The data is partitioned into five folds, each fold is divided into training, validation and test set. The training set forms 80% of the dataset, and the validation and test sets forms 10% each. This process is

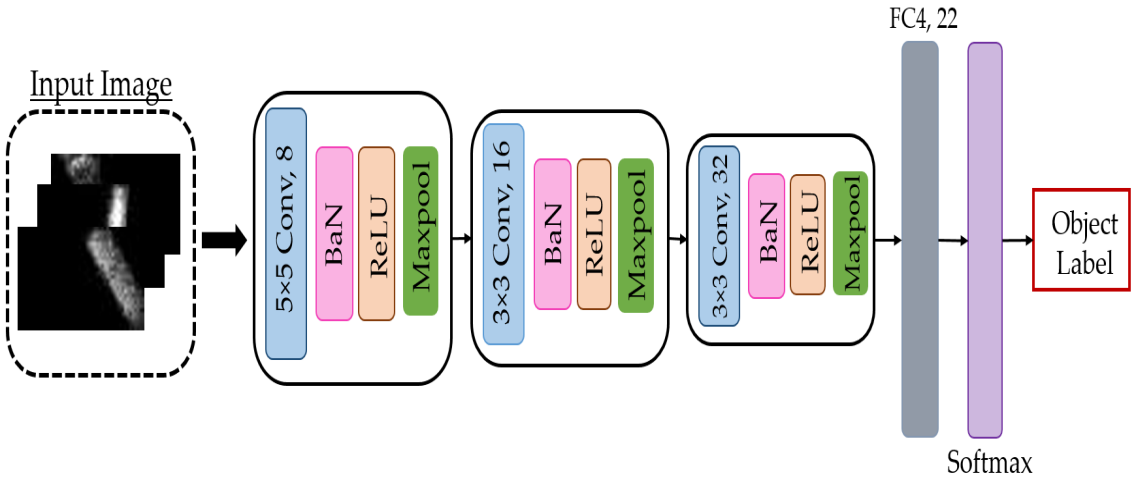


Fig. 4.2 Architecture of the tested model

then repeated five times until all the folds are used, without having common elements across all folds for validation and test set as shown in Figure 4.3 .

For each training process, the training set is composed of 880 images, 40 images for each label, whilst each of the validation and test set is composed of 110 images. Training the model from scratch requires a large dataset to achieve high accuracy. For that reason, data augmentation techniques i.e. flipping, rotation, and translation in the X and Y axis have been applied to the dataset. Hence, the amount of tactile data available for training and validation is increased to 5280, and 660 respectively. The performance of the implemented model is evaluated based on the recognition rates achieved in a classification experiment of the test set composed of 110 original images(objects) from 22 classes.

For embedded applications, with computational, memory and energy constraints, it is necessary to decrease the number of trainable parameters in the CNN model. In this work, we chose to decrease the number of parameters of the trained model by decreasing the input image size (i.e. lower resolution images), an example is shown in Figure 4.4. For that reason, several experiments have been done to choose the smaller size of the input data keeping the same classification accuracy. The input shapes were chosen in a way that each shape induces

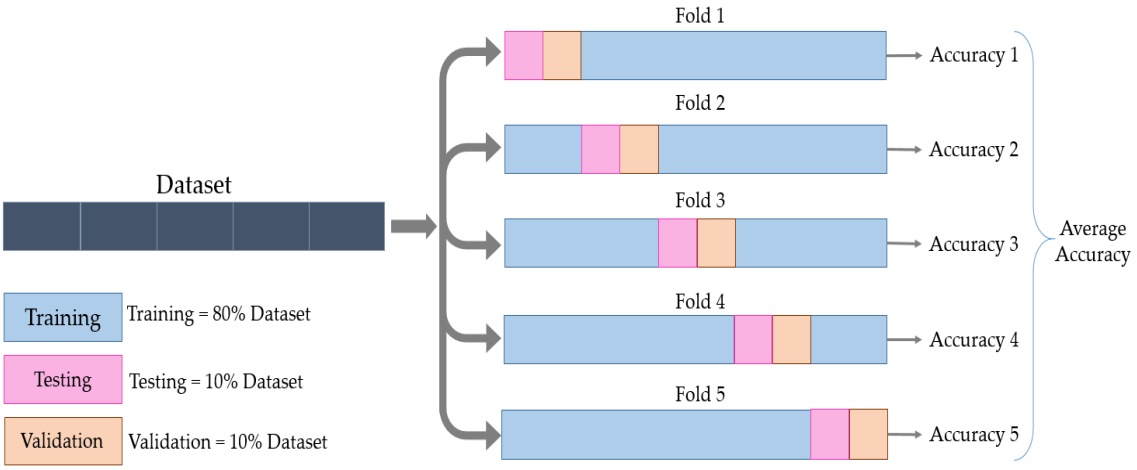


Fig. 4.3 Visual Representation of Train, Test, and validation split using cross validation

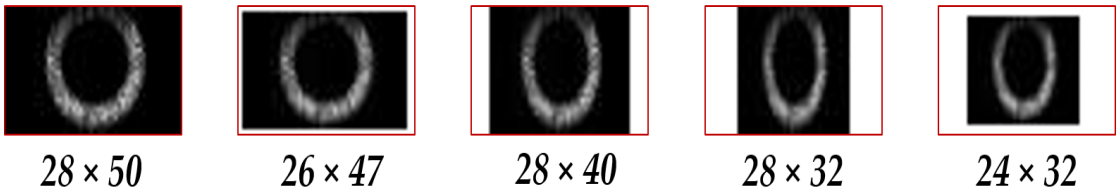


Fig. 4.4 Example of image resize for a "sticky tape" object, the red canvas is drawn for illustration, which illustrates the original image size (28×50).

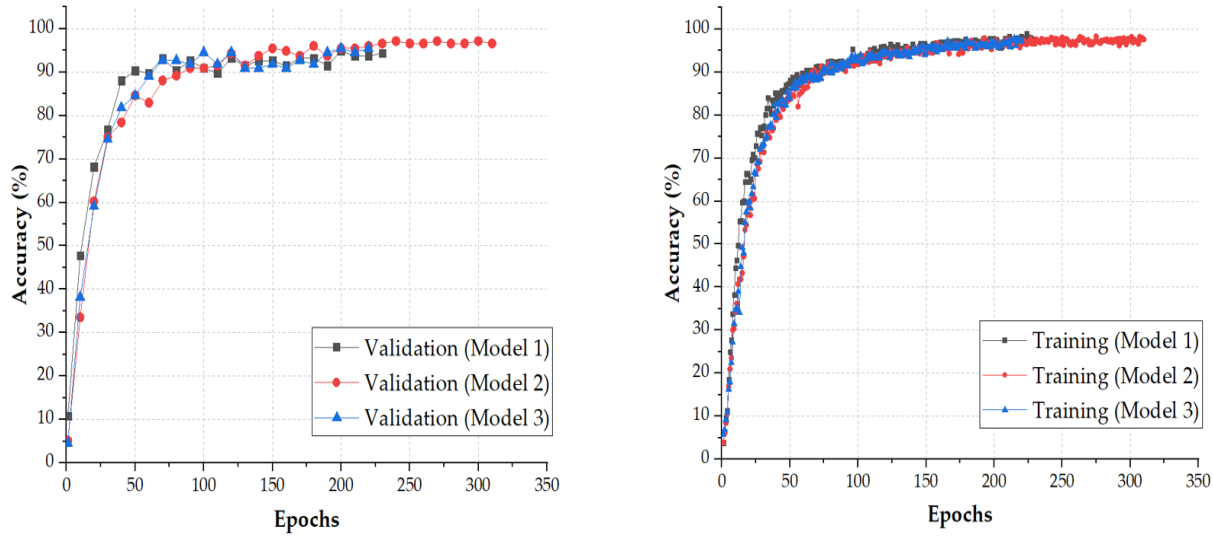
a reduction in the number of parameters. This number is obtained using $tf.model.summary()$ method from TensorFlow [113].

Table 4.1 shows how the number of parameters of the layers depends on the input shape. The change in the input shape affects only the number of parameters of the fully connected layer. This is due to the fact that the number of parameters in the Convolutional layer depends only on the size and number of the filters assigned for each layer $((width\ of\ the\ filter \times height\ of\ the\ filter) + 1) \times no.of filters)$, while in the FC layer the number of parameters $(no.of\ neurons\ in\ FC\ layer\ \times (no.\ of\ neurons\ in\ previous\ layer + 1))$ is affected by the size of the input image and the output layer. The performance of the model was studied with five different input shapes shown in Table 4.1. Thus resulting five different model with different input shapes, each one trained from scratch 5 times (one time per fold), which outputs 25 trained NNs. Figure 4.5 shows the training and validation accuracy over epochs,for the first

Table 4.1 Distribution of number of parameters on the models' layers

Layers	Model 1 (28×50)	Model 2 (26×47)	Model 3 (28×40)	Model 4 (28×32)	Model 5 (24×32)
Conv1	208	208	208	208	208
BaN1	16	16	16	16	16
Conv2	1168	1168	1168	1168	1168
BaN2	32	32	32	32	32
Conv3	4640	4640	4640	4640	4640
BaN3	64	64	64	64	64
FC	19734	16918	14102	11286	8470
Total	25862	23046	20230	17414	14598

three models among the five models. The figure shows that the accuracy achieved of the three models is close to 100%. Each model was evaluated on Matlab by running a classification task on the test set.



(a) (b)

Fig. 4.5 Learning accuracy for the 3 configurations of the TactNet4 model: (a)Training, (b) Validation

Figure 4.6 shows the change in the number of trainable parameters and the average classification accuracy, with respect to the change in the input shape as well as the FLOPs. The classification accuracy presents the average test accuracy among the five folds. The

figure shows that it is possible to decrease the input size from 28×50×1 to 26×47×1 or to 28×40×1 and achieve an increase in the classification accuracy from 90.70% to 91.98% and 90.88% respectively. Decreasing the input size of the model results in a drop in the trainable parameters from 25 862 to 23 046, and 20 230 parameters respectively for the aforementioned models. This decrease in number of parameters, will also induce a decrease of number of Floating Point Operations (FLOPs) as shown in Figure 4.6, the average ratio of the decrease of number of parameters with respect to the decrease in number of FLOPs is 1/44 i.e with each decrease of number of parameters, there is 44 times decrease of FLOPs. The number of FLOPs in Figure 4.6, corresponds to Convolutional layers only where resides most of the FLOPs, these FLOPs are calculated according to the following formula [114]: $FLOPs = n \times m \times k$, where n is the number of kernels, k is the size of the kernel ($width \times height \times depth$) and m the size of output feature map ($width \times height$), the depth in the kernel size corresponds to the depth of the input feature map.

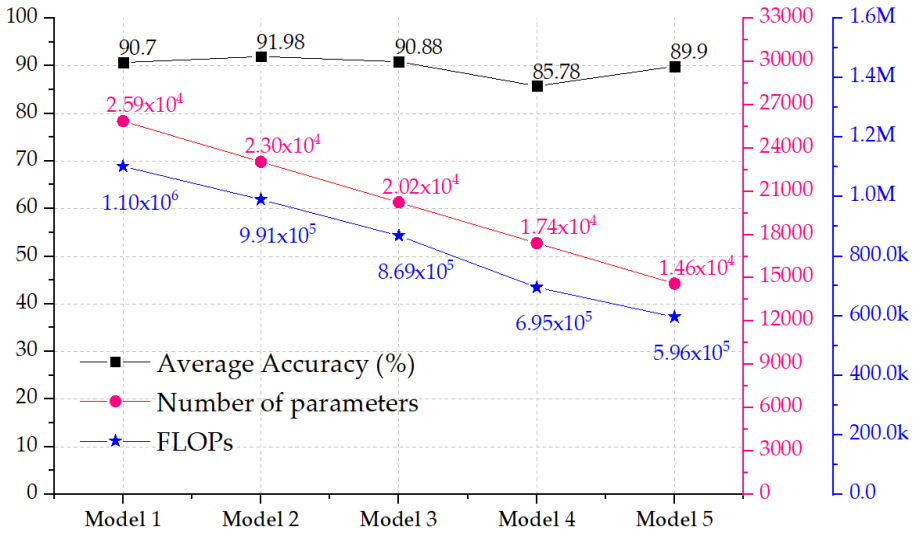


Fig. 4.6 Comparison of the performance, number of trainable parameters, and FLOPs in Convolution layers.

4.4 Embedded Hardware Implementations

The models obtained from Matlab, are converted to Open Neural Network Exchange (ONNX) format [115]. ONNX provides an open source format for AI models, both deep learning and traditional ML, which enables the inter-operability between different frameworks. Figure 4.7. shows how the CNN model is converted into different formats for different hardware platforms. Figure 4.6 shows the number of trainable parameters and the corresponding accuracy for each model. It is clearly shown that all models preserve comparable accuracy, but the best are the first three i.e. Model 1, Model 2, and Model 3. However, since Model 2 and Model 3 have demonstrated a reduced number of training parameters and accordingly a reduced number of operations (FLOPs), they have been selected for the hardware implementation. This choice is based on the fact that reducing FLOPs reduces the inference time and power consumption.

The reason behind the selection of hardware platforms:

1. Custom architecture targeting embedded implementation of Neural Networks e.g. Movidius NCS2.

2. High usability of ARM processors in embedded architectures e.g. Raspberry Pi 4.

3. High performance architecture, designed for parallel processing in general, and also optimised for embedded applications: e.g. NVIDIA Jetson TX2 .

4. Support for execution of pre-trained Neural Network models coming from different platforms without retraining.

4.4.1 Movidius Neural Compute Stick2 (NCS2)

Movidius NCS2 is a hardware accelerator designed by Intel for on-chip neural network inference especially CNNs, equipped with Intel Movidius MyriadX Vision Processing Unit (VPU), it contains 16 SHAVE cores (Streaming Hybrid Architecture Vector Engine) [116],

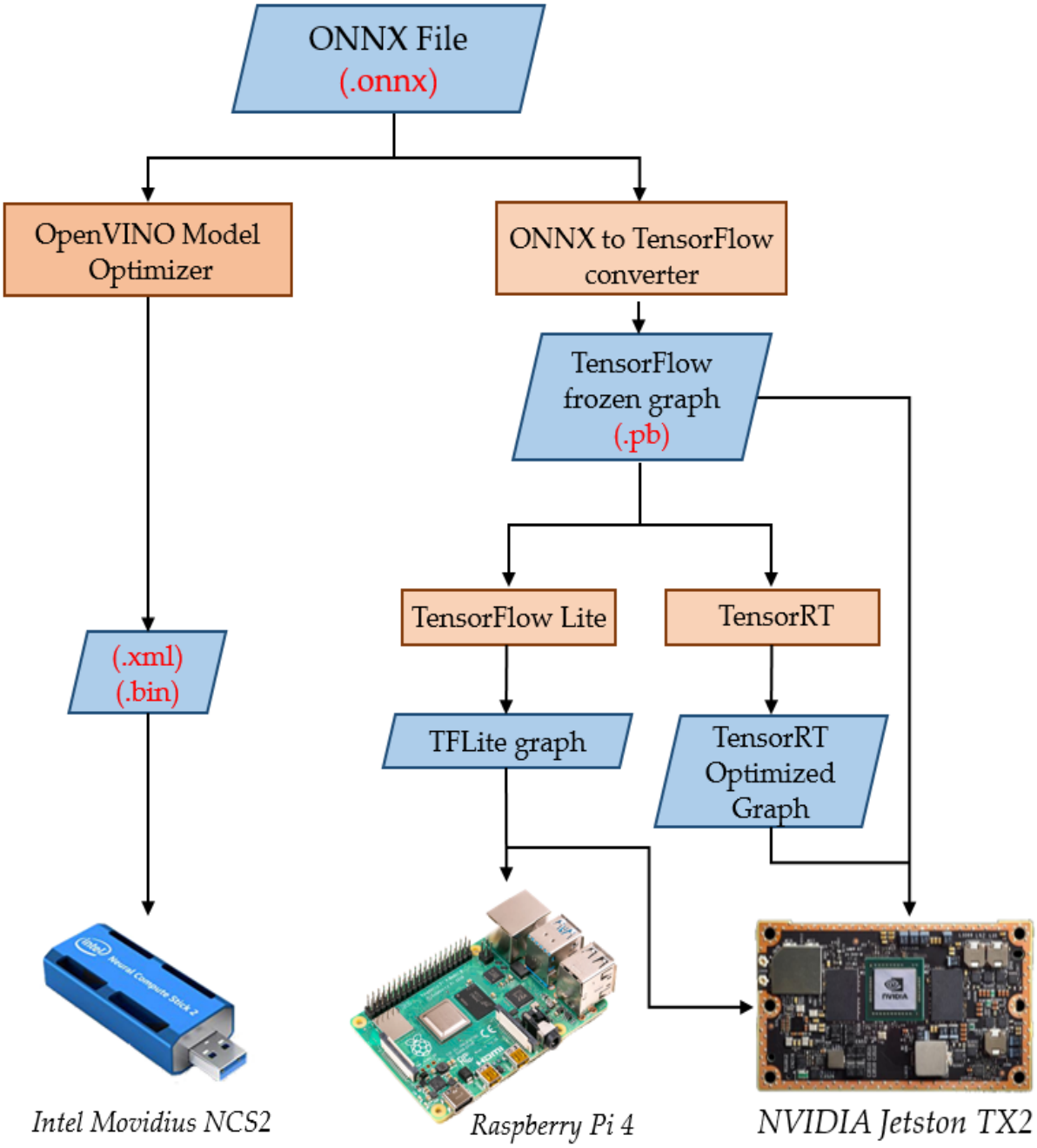


Fig. 4.7 Implementation Flow

and a dedicated hardware neural network accelerator. It requires a host to flash the neural network, and also to feed it with data and invoke the inference to get the results back via the USB 3.0 port. The host can be a Linux, Windows or Mac Based machine. To achieve these tasks, Intel provides OpenVINO: Open Visual Inference and Neural network Optimisation Toolkit, a cross platform toolkit that enables deep learning inference and easy heterogeneous execution across multiple Intel® hardware (VPU, GPU, CPU, FPGA). Optimisation offered by OpenVINO are: Batch-Normalisation and Scale-Shift, linear operations merge and linear operations fusion. Details are mentioned in [117].

4.4.2 Jetson TX2

NVIDIA's Jetson TX2 [118] is a power-efficient embedded AI computing device, designed mainly for edge AI, belongs to Pascal™-family GPU, loaded with 8 GB of memory and 59.7 GB/s of memory bandwidth. In this experiment we used TensorFlow [113] for the inference, as well as NVIDIA TensorRT [119] under Ubuntu OS. TensorFlow is an open source end-to-end machine learning platform, while TensorRT is a platform for high-performance deep learning inference dedicated for NVIDIA hardware, It includes a deep learning inference optimizer and a runtime that delivers low latency and high-throughput for deep learning inference applications.

As an optimisation for TensorFlow, TensorFlow Lite (TFLite) [120] is an open source deep learning framework for on-device inference. The same TensorFlow model can be converted into TFLite model. To perform an inference with a TFLite model, The TFLite interpreter is required, which uses a static graph ordering and a custom (less-dynamic) memory allocator to ensure minimal load, initialization, and execution latency [120], also reducing weights' precision e.g. floating point, vs fixed point precision without affecting the accuracy.

4.4.3 ARM

As for the implementation on ARM architecture, we used Raspberry Pi 4, equipped with a Quad core Cortex-A72 (ARM v8) 64-bit System on Chip (SoC) @ 1.5 GHz and 4 GB RAM. For the inference on this hardware, we used TFLite runtime library(interpreter), under Ubuntu OS.

For all the mentioned platforms, both power consumption and inference time were calculated. The inference time was calculated by averaging 110 inferences, which correspond to the test set size. As for the power consumption, two methods were used:

1. Using provided APIs in Jetson TX2, which provides readings about voltage, power, and input current to the GPU.

2. By using external USB multimeter, connected in serial to the power source for both Raspberry Pi, and the Movidius NCS2.

Table 4.2 Accuracy results for 10 runs on Model 2, Fold 4

Trials	**Accuracy (%)**
1	96.36
2	92.73
3	94.55
4	91.82
5	97.27
6	93.64
7	92.73
8	95.45
9	96.36
10	92.73
Average $\pm$ Stdev	94.36 $\pm$ 1.904 %

4.5 Results And Discussion

In this work, we did achieve a better accuracy in tactile data classification using CNN, compared to the original model obtained in [86], even by resizing the input therefore decreasing the number of trainable parameters.The chosen models reduced the number of trainable parameters by a maximum of 21.77% of original trainable parameters, and also increased the accuracy by a maximum of 1.28%, noting that Model5 ($24\times32\times1$) with 0.8% less accuracy than the original model, has 42% less trainable parameters. Choosing the right model depends on the implementation, i.e. a trade-off between accuracy and hardware complexity should take place: if the best accuracy is targeted then Model 2 should be selected; while the choice of Model 3 would be when less hardware complexity is needed but with a small accuracy degradation. Reducing the input size while still keeping the same, or even better accuracy can be explained in three points:

1. The random initialization of the weights may lead in different runs to different accuracy results, e.g. 10 different runs for training the fold 4 of Model 2, with same hyper-parameters give different results as shown in Table 4.2, which shows an average of 94.36% and a standard deviation of 1.904%.

2. Random selection of batch data, and data shuffling will affect also the update of the weights and make them different from a training to another.

3. Features extraction process achieved by CNN is error-resilient [121]. A CNN can still extract features even with some manipulation of the input image. This is one of the reasons of data augmentation [122] when training neural networks, which is to let the neural network learn the features even from augmented images (scaled, rotated, flipped, etc..) instead of learning only the samples in the original dataset . In our case the features are still detectable even after image resize, as shown in Figure 4.4.

According to Table 4.3 and Table 4.4, the smallest power consumption and inference time were obtained using TensorRT under Jetson TX2, which is 153 mW dynamic power

Table 4.3 Comparison of the inference time between models

Platform		Inference time (ms)		
Hardware	Software	Model 1	Model 2	Model 3
Jetson TX2	TensorRT	5.5597	5.2905	5.919
	TensorFlow	6.2943	5.4691	5.946
	TFLite	1.3384	1.2181	1.2445
Core i7	Matlab	3.245	2.6139	2.4715
Movidius NCS2	OpenVINO	1.9	1.9	1.86
Raspberry Pi4	TFLite	1.615	1.473	1.21

within 5.29 ms as inference time, implies $0.809e^{-3}$ Joules dynamic energy. While the most dynamic energy consumption was for the Intel Movidius NCS2, 1.9 ms$\times$800 mW =$1.52e^{-3}$ Joules as shown in Table 4.5. Regarding the power consumption, since the neural network used, is small compared to the hardware capacity, the power consumption was almost the same for the three models, noting that the accuracy on the USB power meter, is on 10 mW scale, so that a difference less than 10 mW between two measurements, cannot be detected using this instrument.

Table 4.4 Power consumption

Platform		Current (mA)		Voltage (V)	Consumed Power (mW)		
Hardware	Software	Static	Total		Static	Total	Dynamic
Jetson	TensorRT	8	16	19.072	152	305	153
	TensorFlow	8	16	19.072	152	305	153
Movidius NCS2	OpenVINO	-	160	5	-	800	800
Raspberry Pi4	TFLite	560	700	5	2800	3500	700

Table 4.5 Energy consumption

Platform		Energy consumption (µJ)		
Hardware	Software	Model 1	Model 2	Model 3
Jetson TX2	TensorRT	850.6341	809.4465	905.607
	TensorFlow	963.0279	836.7723	909.738
Movidius NCS2	OpenVINO	1520	1520	1488
Raspberry Pi4	TFLite	1130.5	1031.1	847

4.6 Conclusions

This chapter presented the implementation of smart tactile sensing system based on embedded CNN approach. The proposed model has optimized a state of art model proposed in [86] by reducing the input data size. Experimental results have shown comparable results in terms of accuracy after reducing the size from (28×50) to (26×47) and (28×40). The hardware implementation on different hardware platforms namely Movidius NCS2, NVIDIA's Jetson TX2, and Cortex-A72 (ARM v8) have been provided. The proposed models have shown better performance on embedded hardware platforms when time inference has been compared. Power consumption has also been measured and compared among different platforms. Targeting portable tactile sensing systems, the proposed work has demonstrated the feasibility of integrating machine learning methods on embedded hardware platform to enable intelligence for such system. This may pave the way for the smart tactile sensing systems to be applied in prosthetics and robotics.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we have proposed different AI based solutions for tactile data processing in an embedded environment. The focus was on deep learning, with a feasibility study concerning the power consumption and latency on different embedded commercial hardware platforms e.g. Jetson TX2, Intel Neural Compute Stick 2 and Raspberry PI. Two case studies were selected, one on touch modality classification, and the other on tactile object recognition. The main difference between these two case studies, is the sensor size and data type i.e. static vs dynamic. In the tactile object recognition problem mentioned in Chapter 4, the sensory array is large enough (28×50) to give a heat-map for an object, and static data is considered, while in the touch modality classification problem, data is dynamic (tensorial), and the sensory array was small (4×4).

To solve the touch modality classification, two approaches were considered, one is to transform tensorial data into static data and then apply traditional CNNs trained on natural images through transfer learning into synthetic tactile images, and the other is to employ solutions where time is considered as a dimension e.g. Recursive Neural Networks, as shown in Chapter 3, after proposing a data organisation and subsampling that reduced -together with

the proposed solution- the FLOPs up to 99.98% and the model size by 89.3% with respect to SoA solutions on the same dataset, as well as a high accuracy up to 84.23%.

One important property in this solution, is 1) the processing can start once a single reading from the sensor array is ready, which was not present in the previous SoA solutions for this problem, and 2) a parallelism between the data reading and processing can be done by a simple pipeline as explained in Figure 3.1.c. On the other hand, it is proven that it is possible to achieve a real time tactile application with a latency less than 2ms, and an energy consumption of less than 1000 µJ as shown in Table 4.5. For more practical application, a tactile system equipped with a 1000 mAh, 5 V battery (two times less power than most smartphone batteries), and a Jetson TX2 GPU device executing the Model3 of the tactile object recognition mentioned in Chapter 4. The time duration this power bank can last, is calculated according to the following equations:

$$TotalEnergyPowerbank(TEpb) = 1000\,\text{mAh} \times 5\,\text{V} = 5000\,\text{mWh} = 5\,\text{Wh} = 5 \times 3600\,\text{J} = 18\,000\,\text{J} \tag{5.1}$$

$$TotalLifeTime = TEpb/SingleInferenceTotalEnergy^{*} = 18\,000\,\text{J}/1000\,\mu\text{J} = 18 \times 10^{6}\ inferences \tag{5.2}$$

*The total energy per inference is less than 1000 J , we considered this number for simplification. If we consider that the system consumes at worst in sleep time, the same energy consumed while executing the inference. The system will last for 5.92 ms $\times 18 \times 10^{6} = 106\,560$ s = 29.6 h.

5.2 Directions for Future Research

Having chosen an effective algorithm for touch modality classification, the final goal is:

1. To embed the LSTM/GRU based solution on a custom-made circuit, including the tactile sensors and the interface-electronics, to be embodied in the prosthetic or other tactile-based solutions. This circuit will endorse up to 16 sensors, Data Acquisition, Analog-to-Digital conversion, and the proposed algorithm, the pre-processing will be done on hardware also. This work has already started with an [123] architecture, the proposed solution in Chapter 3 is suitable for such interface, because it can fit easily within its memory and computation limits.

2. In a further step, energy-efficient techniques introduced in Subsection 1.2.2 should be applied in order to achieve more reduction in the power consumption.

3. Different problems should be addressed, not only touch modality classification, in order to mimic as much as possible the functionality of real human-skin.

Finally, a critical point concerning all the aforementioned solutions for touch modality classification, is that all the proposed solutions require all the data to be ready in order to give a decision (i.e. a classification output), while in real human-skin, people are able to identify objects sometime by a single touch, and sometime by palpation or continuous touch. This kind of decision should be supported by tactile skin processing algorithm, where at each instant T a local decision can be made, and at each time interval $[T, T + interval]$ a cumulative global decision is done as well. The cost of pre-processing should be studied as well.

References

[1] Ravinder S. Dahiya, Philipp Mittendorfer, Maurizio Valle, Gordon Cheng, and Vladimir J. Lumelsky. Directions Toward Effective Utilization of Tactile Skin: A Review. *IEEE Sensors Journal*, 13(11):4121–4138, November 2013.

[2] G. Cheng, E. Dean-Leon, F. Bergner, J. R. G. Olvera, Q. Leboutet, and P. Mittendorfer. A Comprehensive Realization of Robot Skin: Sensors, Sensing, Control, and Applications. *Proceedings of the IEEE*, pages 1–18, 2019.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[4] Marta Franceschi, Lucia Seminara, Strahinja Dosen, Luigi Pinna, H Fares, Moustafa Saleh, Maurizio Valle, and Dario Farina. Live demonstration: Electrotactile feedback from an electronic skin through flexible electrode matrix. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–1. IEEE, 2018.

[5] Mohamad Alameh, Moustafa Saleh, Flavio Ansovini, Hoda Fares, Ali Ibrahim, Marta Franceschi, Lucia Seminara, Maurizio Valle, Strahinja Dosen, and Dario Farina. Live demonstration: System based on electronic skin and cutaneous electrostimulation for sensory feedback in prosthetics. In *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–1. IEEE, 2018.

[6] M. Saleh, A. Ibrahim, F. Ansovini, Y. Mohanna, and M. Valle. Wearable system for sensory substitution for prosthetics. In *2018 New Generation of CAS (NGCAS)*, pages 110–113, 2018.

[7] Moustafa Saleh, Yahya Abbass, Ali Ibrahim, and Maurizio Valle. Experimental assessment of the interface electronic system for pvdf-based piezoelectric tactile sensors. *Sensors*, 19(20):4437, 2019.

[8] J. Russell Stuart and Peter Norvig. *Artificial intelligence: a modern approach*. Prentice Hall, 2009.

[9] David Poole, Alan Mackworth, and Randy Goebel. Computational Intelligence. 1998.

[10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995. Company: Springer Distributor: Springer Institution: Springer Label: Springer Number: 3 Publisher: Kluwer Academic Publishers.

[11] TM Cover and P Hart. Nearest neighbor decision rule. In *IEEE Transactions on Information Theory*, volume 12,2, pages 272–+. IEEE-INST ELECTRICAL ELECTRONICS ENGINEERS INC 345 E 47TH ST, NEW YORK, NY ..., 1966.

[12] S. A. Dudani. The Distance-Weighted k-Nearest-Neighbor Rule. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(4):325–327, April 1976. Conference Name: IEEE Transactions on Systems, Man, and Cybernetics.

[13] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[14] S. R. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3):660–674, 1991.

[15] Daniel Svozil, Vladimír Kvasnicka, and Jiri Pospichal. "introduction to multi-layer feed-forward neural networks". *Chemometrics and Intelligent Laboratory Systems*, 39(1):43 – 62, 1997.

[16] Thomas P Vogl, JK Mangis, AK Rigler, WT Zink, and DL Alkon. Accelerating the convergence of the back-propagation method. *Biological cybernetics*, 59(4-5):257–263, 1988.

[17] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.

[18] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, and Jianfei Cai. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018. Publisher: Elsevier.

[19] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[20] Jon Gauthier. Conditional generative adversarial nets for convolutional face generation. *Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition, Winter semester*, 2014(5):2, 2014.

[21] Xinyuan Chen, Chang Xu, Xiaokang Yang, Li Song, and Dacheng Tao. Gated-gan: Adversarial gated networks for multi-collection style transfer. *IEEE Transactions on Image Processing*, 28(2):546–560, 2018.

[22] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392*, 2019.

[23] Lorenzo Cunial, Ahmet Erdem, Cristina Silvano, Mirko Falchetto, Andrea C Ornstein, Emanuele Plebani, Giuseppe Desoli, and Danilo Pau. Parallelized convolutions for embedded ultra low power deep learning soc. In *2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI)*, pages 1–4. IEEE, 2018.

[24] Lei Clifton, David A Clifton, Marco AF Pimentel, Peter J Watkinson, and Lionel Tarassenko. Gaussian process regression in vital-sign early warning systems. In *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 6161–6164. IEEE, 2012.

[25] Zidong Du, Krishna Palem, Avinash Lingamneni, Olivier Temam, Yunji Chen, and Chengyong Wu. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. In *2014 19th Asia and South Pacific design automation conference (ASP-DAC)*, pages 201–206. IEEE, 2014.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, May 2017.

[27] Ali Ibrahim and Maurizio Valle. Real-time embedded machine learning for tensorial tactile data processing. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(11):3897–3906, 2018.

[28] Boris Murmann, Daniel Bankman, Elaina Chai, Daisuke Miyashita, and Lita Yang. Mixed-signal circuits for embedded machine-learning applications. In *2015 49th Asilomar conference on signals, systems and computers*, pages 1341–1345. IEEE, 2015.

[29] Vivienne Sze. Designing hardware for machine learning: The important role played by circuit designers. *IEEE Solid-State Circuits Magazine*, 9(4):46–54, 2017.

[30] Mario Osta, Mohamad Alameh, Hamoud Younes, Ali Ibrahim, and Maurizio Valle. Energy efficient implementation of machine learning algorithms on hardware platforms. In *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 21–24. IEEE, 2019.

[31] Simone Benatti, Fabio Montagna, Victor Kartsch, Abbas Rahimi, Davide Rossi, and Luca Benini. Online learning and classification of emg-based gestures on a parallel ultra-low power platform using hyperdimensional computing. *IEEE transactions on biomedical circuits and systems*, 13(3):516–528, 2019.

[32] M. Osta, A. Ibrahim, M. Magno, M. Eggimann, A. Pullini, P. Gastaldo, and M. Valle. An Energy Efficient System for Touch Modality Classification in Electronic Skin Applications. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 2019-May, pages 1–4. IEEE, may 2019.

[33] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Computer Architecture News*, 44(3):367–379, 2016.

[34] Rafał Kułaga and Marek Gorgoń. Fpga implementation of decision trees and tree ensembles for character recognition in vivado hls. *Image Processing & Communications*, 19(2-3):71–82, 2014.

[35] Shereen Afifi, Hamid GholamHosseini, and Roopak Sinha. A low-cost fpga-based svm classifier for melanoma detection. In *2016 IEEE EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, pages 631–636. IEEE, 2016.

[36] Yuliang Pu, Jun Peng, Letian Huang, and John Chen. An efficient knn algorithm implemented on fpga based heterogeneous computing system using opencl. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 167–170. IEEE, 2015.

[37] Dongjoo Shin, Jinmook Lee, Jinsu Lee, Juhyoung Lee, and Hoi-Jun Yoo. An energy-efficient deep learning processor with heterogeneous multi-core architecture for convolutional neural networks and recurrent neural networks. In *2017 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, pages 1–2. IEEE, 2017.

[38] Bert Moons, Bert De Brabandere, Luc Van Gool, and Marian Verhelst. Energy-efficient convnets through approximate computing. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–8. IEEE, 2016.

[39] Syed Shakib Sarwar, Gopalakrishnan Srinivasan, Bing Han, Parami Wijesinghe, Akhilesh Jaiswal, Priyadarshini Panda, Anand Raghunathan, and Kaushik Roy. Energy efficient neural computing: A study of cross-layer approximations. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(4):796–809, 2018.

[40] Lukas Cavigelli and Luca Benini. Origami: A 803-gop/s/w convolutional network accelerator. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(11):2461–2475, 2016.

[41] Bert Moons, Roel Uytterhoeven, Wim Dehaene, and Marian Verhelst. 14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 246–247. IEEE, 2017.

[42] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5687–5695, 2017.

[43] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]*, April 2017. arXiv: 1704.04861.

[44] Dongyoung Kim, Junwhan Ahn, and Sungjoo Yoo. A novel zero weight/activation-aware hardware architecture of convolutional neural network. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1462–1467. IEEE, 2017.

[45] Ali Ibrahim, Mario Osta, Mohamad Alameh, Moustafa Saleh, Hussein Chible, and Maurizio Valle. Approximate computing methods for embedded machine learning. In *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 845–848. IEEE, 2018.

[46] Mohamed M Sabry Aly, Tony F Wu, Andrew Bartolo, Yash H Malviya, William Hwang, Gage Hills, Igor Markov, Mary Wootters, Max M Shulaker, H-S Philip Wong, et al. The n3xt approach to energy-efficient abundant-data computing. *Proceedings of the IEEE*, 107(1):19–48, 2018.

[47] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. Survey and Benchmarking of Machine Learning Accelerators. *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9, September 2019. arXiv: 1908.11348.

[48] Mohamad Alameh, Ali Ibrahim, Maurizio Valle, and Gabriele Moser. DCNN for Tactile Sensory Data Classification based on Transfer Learning. In *2019 15th Conference on Ph.D Research in Microelectronics and Electronics (PRIME)*, pages 237–240, July 2019.

[49] Mohamad Alameh, Yahya Abbass, Ali Ibrahim, and Maurizio Valle. Smart Tactile Sensing Systems Based on Embedded CNN Implementations. *Micromachines*, 11(1):103, 2020.

[50] H. Fares, L. Seminara, A. Ibrahim, M. Franceschi, L. Pinna, M. Valle, S. Dosen, and D. Farina. Distributed Sensing and Stimulation Systems for Sense of Touch Restoration in Prosthetics. In *2017 New Generation of CAS (NGCAS)*, pages 177–180, September 2017.

[51] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[52] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.

[53] Dan Cireşan, Ueli Meier, and Juergen Schmidhuber. Multi-column Deep Neural Networks for Image Classification. *arXiv:1202.2745 [cs]*, February 2012. arXiv: 1202.2745.

[54] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. *arXiv:1310.1531 [cs]*, October 2013. arXiv: 1310.1531.

[55] K. Fukushima, S. Miyake, and T. Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):826–834, September 1983.

[56] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, December 1989.

[57] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[58] L. Mou, X. Zhu, M. Vakalopoulou, K. Karantzalos, N. Paragios, B. Le Saux, G. Moser, and D. Tuia. Multitemporal very high resolution from space: Outcome of the 2016 IEEE GRSS Data Fusion Contest. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(8):3435–3447, 2017.

[59] Mahdi Rasouli, Yi Chen, Arindam Basu, Sunil L. Kukreja, and Nitish V. Thakor. An Extreme Learning Machine-Based Neuromorphic Tactile Sensing System for Texture Recognition. *IEEE transactions on biomedical circuits and systems*, 12(2):313–325, 2018.

[60] Huaping Liu, Jie Qin, Fuchun Sun, and Di Guo. Extreme kernel sparse learning for tactile object recognition. *IEEE transactions on cybernetics*, 47(12):4509–4520, 2017.

[61] Abdul Md Mazid and ABM Shawkat Ali. Grasping force estimation detecting slip by tactile sensor adopting machine learning techniques. In *TENCON 2008-2008 IEEE Region 10 Conference*, pages 1–6. IEEE, 2008.

[62] David Silvera Tawil, David Rye, and Mari Velonaki. Interpretation of the modality of touch on an artificial arm covered with an EIT-based sensitive skin. *The International Journal of Robotics Research*, 31(13):1627–1641, November 2012.

[63] Juan M. Gandarias, Jesus M. Gomez-de Gabriel, and Alfonso Garcia-Cerezo. Human and object recognition with a high-resolution tactile sensor. In *2017 IEEE SENSORS*, pages 1–3, Glasgow, October 2017. IEEE.

[64] Wenzhen Yuan, Yuchen Mo, Shaoxiong Wang, and Edward H Adelson. Active clothing material perception using tactile sensing and deep learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[65] Mohsen Kaboli, Alex Long, and Gordon Cheng. Humanoids learn touch modalities identification via multi-modal robotic skin and robust tactile descriptors. *Advanced Robotics*, 29(21):1411–1425, November 2015.

[66] Paolo Gastaldo, Luigi Pinna, Lucia Seminara, Maurizio Valle, and Rodolfo Zunino. Computational intelligence techniques for tactile sensing systems. *Sensors (Switzerland)*, 14(6):10952–10976, 2014.

[67] Imagenet. http://www.image-net.org. [Online]; Accessed:2020-11-21.

[68] NVIDIA Jetson Modules and Developer Kits for Embedded Systems Development.

[69] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. arXiv: 1512.03385.

[70] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity Mappings in Deep Residual Networks. *arXiv:1603.05027 [cs]*, March 2016. arXiv: 1603.05027.

[71] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. *arXiv:1512.00567 [cs]*, December 2015. arXiv: 1512.00567.

[72] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *arXiv:1602.07261 [cs]*, February 2016. arXiv: 1602.07261.

[73] Kevin Kiningham, Michael Graczyk, and Athul Ramkumar. Design and Analysis of a Hardware CNN Accelerator. *Small: nano micro*, 27(6):8, 2016.

[74] Marta Franceschi, Lucia Seminara, Strahinja Dosen, Matija Strbac, Maurizio Valle, and Dario Farina. A system for electrotactile feedback using electronic skin and flexible matrix electrodes: Experimental evaluation. *IEEE transactions on haptics*, 10(2):162–172, 2016.

[75] Ravinder Dahiya, Nivasan Yogeswaran, Fengyuan Liu, Libu Manjakkal, Etienne Burdet, Vincent Hayward, and Henrik Jörntell. Large-area soft e-skin: the challenges beyond sensor designs. *Proceedings of the IEEE*, 107(10):2016–2033, 2019.

[76] Ji Young Lee and Franck Dernoncourt. Sequential short-text classification with recurrent and convolutional neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 515–520, San Diego, California, June 2016. Association for Computational Linguistics.

[77] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*, 2017.

[78] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[79] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 1724–1734, 2014.

[80] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[81] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[82] Vladimir N Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

[83] Ghazal Rouhafzay and Ana-Maria Cretu. An Application of Deep Learning to Tactile Data for Object Recognition under Visual Guidance. *Sensors*, 19(7):1534, January 2019.

[84] Zineb Abderrahmane, Gowrishankar Ganesh, André Crosnier, and Andrea Cherubini. Visuo-Tactile Recognition of Daily-Life Objects Never Seen or Touched Before. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1765–1770. IEEE, 2018.

[85] Chia Hsien Lin, Todd W. Erickson, Jeremy A. Fishel, Nicholas Wettels, and Gerald E. Loeb. Signal processing and fabrication of a biomimetic tactile sensor array with thermal, force and microvibration modalities. In *2009 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 129–134, December 2009.

[86] Juan M. Gandarias, Alfonso J. Garcia-Cerezo, and Jesus M. Gomez-de Gabriel. CNN-Based Methods for Object Recognition With High-Resolution Tactile Sensors. *IEEE Sensors Journal*, 19(16):6872–6882, 2019.

[87] Jianhua Li, Siyuan Dong, and Edward Adelson. Slip detection with combined tactile and visual information. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7772–7777. IEEE, 2018.

[88] Haoying Wu, Daimin Jiang, and Hao Gao. Tactile motion recognition with convolutional neural networks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1572–1577. IEEE, 2017.

[89] Jennifer Kwiatkowski, Deen Cockburn, and Vincent Duchaine. Grasp stability assessment through the fusion of proprioception and tactile signals using convolutional neural networks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 286–292, Vancouver, BC, September 2017. IEEE.

[90] Francisco Pastor, Juan M Gandarias, Alfonso J García-Cerezo, and Jesús M Gómez-de Gabriel. Using 3d convolutional neural networks for tactile object recognition with robotic palpation. *Sensors*, 19(24):5356, 2019.

[91] V. Sze, Y. Chen, J. Emer, A. Suleiman, and Z. Zhang. Hardware for machine learning: Challenges and opportunities. In *2017 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–8, 2017.

[92] Wenzhen Yuan, Chenzhuo Zhu, Andrew Owens, Mandayam A. Srinivasan, and Edward H. Adelson. Shape-independent hardness estimation using deep learning and a GelSight tactile sensor. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 951–958, May 2017.

[93] Karl Van Wyk and Joe Falco. Slip Detection: Analysis and Calibration of Univariate Tactile Signals. *arXiv:1806.10451 [cs]*, June 2018. arXiv: 1806.10451.

[94] Brayan S. Zapata-Impata, Pablo Gil, and Fernando Torres. Tactile-Driven Grasp Stability and Slip Prediction. *Robotics*, 8(4):85, 2019.

[95] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[96] Siyuan Dong and Alberto Rodríguez. Tactile-based insertion for dense box-packing. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7953–7960, 2019.

[97] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[98] Xingjian Shi, Zhourong Chen, Hao Wang, Dit Yan Yeung, Wai Kin Wong, and Wang Chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *Advances in Neural Information Processing Systems*, 2015-Janua:802–810, 2015.

[99] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. LSTM Fully Convolutional Networks for Time Series Classification. *IEEE Access*, 6:1662–1669, 2017.

[100] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

[101] Sreeraj Rajendran, Wannes Meert, Domenico Giustiniano, Vincent Lenders, and Sofie Pollin. Deep learning models for wireless signal classification with distributed low-cost spectrum sensors. *IEEE Transactions on Cognitive Communications and Networking*, 4(3):433–445, 2018.

[102] Guangyi Zhang, Vandad Davoodnia, Alireza Sepas-Moghaddam, Yaoxue Zhang, and Ali Etemad. Classification of hand movements from eeg using a deep attention-based lstm network. *IEEE Sensors Journal*, 2019.

[103] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[104] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[105] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[106] Ali Ibrahim, Paolo Gastaldo, Hussein Chible, and Maurizio Valle. Real-time digital signal processing based on fpgas for electronic skin implementation. *Sensors*, 17(3):558, 2017.

[107] Uriel Martinez-Hernandez, Tony J. Dodd, and Tony J. Prescott. Feeling the Shape: Active Exploration Behaviors for Object Recognition With a Robotic Hand. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(12):2339–2348, December 2018.

[108] Liang Zou, Chang Ge, Z. Wang, Edmond Cretu, and Xiaoou Li. Novel tactile sensor technology and smart tactile sensing systems: A review. *Sensors*, 17(11):2653, 2017.

[109] Rui Li and Edward H. Adelson. Sensing and Recognizing Surface Textures Using a GelSight Sensor. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1241–1247, Portland, OR, USA, June 2013. IEEE.

[110] Alexander Schmitz, Yusuke Bansho, Kuniaki Noda, Hiroyasu Iwata, Tetsuya Ogata, and Shigeki Sugano. Tactile object recognition using deep learning and dropout. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 1044–1050, November 2014. ISSN: 2164-0572, 2164-0580.

[111] Zineb Abderrahmane, Gowrishankar Ganesh, André Crosnier, and Andrea Cherubini. Haptic zero-shot learning: Recognition of objects never touched before. *Robotics and Autonomous Systems*, 105:11 – 25, 2018.

[112] Klaus Jansen and Hu Zhang. Scheduling malleable tasks. *Handbook of Approximation Algorithms and Metaheuristics*, pages 45–1–45–16, 2007.

[113] TensorFlow. https://www.tensorflow.org. [Online]; Accessed:2020-11-20.

[114] Zongqing Lu, Swati Rallapalli, Kevin Chan, and Thomas La Porta. Modeling the resource requirements of convolutional neural networks on mobile devices. *MM 2017 - Proceedings of the 2017 ACM Multimedia Conference*, pages 1663–1671, 2017.

[115] Open Neural Network Exchange. https://github.com/onnx/onnx/. [Online]; Accessed:2020-11-21.

[116] Intel movidius ncs2. https://software.intel.com/en-us/neural-compute-stick. [Online]; Accessed:2020-11-21.

[117] Openvino model optimization techniques. https://docs.openvinotoolkit.org/latest/_docs_MO_DG_prepare_model_Model_Optimization_Techniques.html. [Online]; Accessed:2020-11-21.

[118] Nvidia jetson modules and developer kits for embedde systems development. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems. [Online]; Accessed:2020-11-21.

[119] NVIDIA TensorRT. https://developer.nvidia.com/tensorrt. [Online]; Accessed:2020-11-21.

[120] TensorFlow Lite. https://www.tensorflow.org/lite. [Online]; Accessed:2020-11-20.

[121] Muhammad Abdullah Hanif, Rehan Hafiz, and Muhammad Shafique. Error resilience analysis for systematically employing approximate computing in convolutional neural networks. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 913–916, March 2018. ISSN: 1558-1101.

[122] Luis Perez and Jason Wang. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. *arXiv:1712.04621 [cs]*, December 2017. arXiv: 1712.04621.

[123] Moustafa Saleh, Ali Ibrahim, Flavio Ansovini, Yasser Mohanna, and Maurizio Valle. Low Power Electronic System for Tactile Sensory Feedback for Prosthetics. 15(1):95–103, 2019.

Appendix A

List of Publications

A.1 Journal Articles

1. **Alameh, Mohamad**, Yahya Abbass, Ali Ibrahim, Gabriele Moser, and Maurizio Valle, "Touch Modality Classification Using Recurrent Neural Networks," in *IEEE Sensors Journal*, doi: 10.1109/JSEN.2021.3055565.
   The article status on 31-01-2021 is "Early Access".

2. Ibrahim, Ali, Hamoud Younes, **Mohamad Alameh**, and Maurizio Valle. "Near Sensors Computation based on Embedded Machine Learning for Electronic Skin," *Procedia Manufacturing* 52 (2020): 295-300.

3. **Alameh, Mohamad**, Yahya Abbass, Ali Ibrahim, and Maurizio Valle. "Smart Tactile Sensing Systems based on Embedded CNN Implementations," *Micromachines* 11, no. 1 (2020): 103.

A.2 Book Chapters

1. Younes, Hamoud, **Mohamad Alameh**, Ali Ibrahim, Mostafa Rizk, and Maurizio Valle. Efficient Algorithms for Embedded Tactile Data Processing. River Publishers, 2020.**(Accepted, not published yet)**

A.3 Conference Papers

1. Osta, Mario, **Mohamad Alameh**, Hamoud Younes, Ali Ibrahim, and Maurizio Valle. "Energy Efficient Implementation of Machine Learning Algorithms on Hardware Platforms." In 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 21-24. IEEE, 2019.

2. **Alameh, Mohamad**, Ali Ibrahim, Maurizio Valle, and Gabriele Moser. "DCNN for Tactile Sensory Data Classification based on Transfer Learning." In 2019 15th Conference on Ph. D Research in Microelectronics and Electronics (PRIME), pp. 237-240. IEEE, 2019.

3. Ibrahim, Ali, Mario Osta, **Mohamad Alameh**, Moustafa Saleh, Hussein Chible, and Maurizio Valle. "Approximate computing methods for embedded machine learning." In 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 845-848. IEEE, 2018.

A.4 Live Demonstration

1. **Alameh, Mohamad**, Moustafa Saleh, Flavio Ansovini, Hoda Fares, Ali Ibrahim, Marta Franceschi, Lucia Seminara, Maurizio Valle, Strahinja Dosen, and Dario Farina. "Live demonstration: System based on electronic skin and cutaneous electrostimulation

for sensory feedback in prosthetics." In 2018 IEEE Biomedical Circuits and Systems Conference (BioCAS), pp. 1-1. IEEE, 2018.

A.5 Statistics

The figure below shows the current impact of the publications, last updated on 17/02/2021.

The updated list can be found on: https://scholar.google.com/citations?hl=en&user=GCFW6osAAAAJ

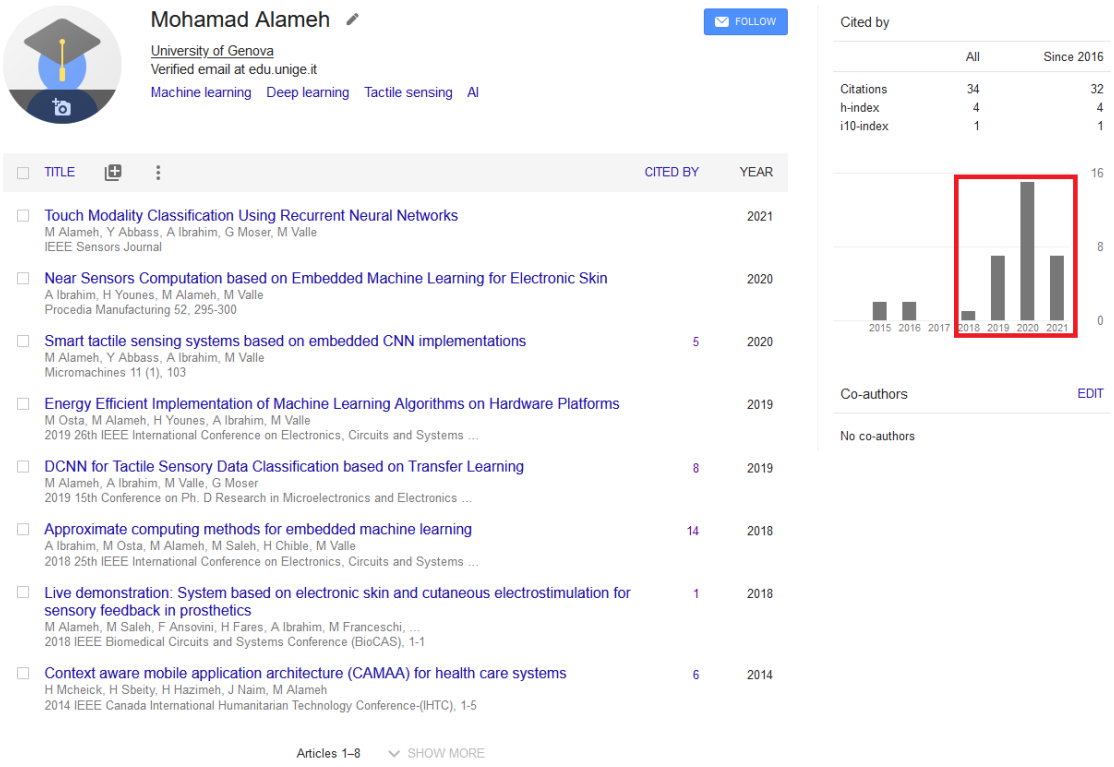


Fig. A.1 List of publications from Google Scholar, last updated 30-01-2021

Appendix B

Touch Classfication Demonstration Codes

B.1 GUI code

```
// FORM3 CODE
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO.Ports;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Devcorp.ColorSpaceSample
{
public partial class Form3 : Form
{
SerialPort sp;
static char AMP = (char)12;
char[] SIM_ON = ">ON<".ToCharArray(); // >T< trigger
char[] SIM_OFF = ">OFF<".ToCharArray();
char[] CMD_SD = ">SD;xxxx<".ToCharArray();
char[] SIM_TRIGGER = ">T<".ToCharArray();
char[] cmdC1 = {'>','C',(char)1,';',
                    AMP,AMP,AMP,AMP,
                    AMP,AMP,AMP,AMP,
                    AMP,AMP,AMP,AMP,
                    AMP,AMP,AMP,AMP,
                    '<'}; // electode1

char[] cmdC2 = {'>','C',(char)2,';',
                    AMP,AMP,AMP,AMP,
                    AMP,AMP,AMP,AMP,
```

```
                    AMP,AMP,AMP,AMP,
                    AMP,AMP,AMP,AMP,
                    '<'}; // electode2

static char AMP_Z = (char)0;
char[] cmdC1_ZERO = {'>','C',(char)1,';',
                        AMP_Z,AMP_Z,AMP_Z,AMP_Z,
                        AMP_Z,AMP_Z,AMP_Z,AMP_Z,
                        AMP_Z,AMP_Z,AMP_Z,AMP_Z,
                        AMP_Z,AMP_Z,AMP_Z,AMP_Z,
                        '<'};
char[] cmdC2_ZERO = {'>','C',(char)2,';',
                        AMP_Z,AMP_Z,AMP_Z,AMP_Z,
                        AMP_Z,AMP_Z,AMP_Z,AMP_Z,
                        AMP_Z,AMP_Z,AMP_Z,AMP_Z,
                        AMP_Z,AMP_Z,AMP_Z,AMP_Z,
                        '<'};

char[] cmdSA = {'>','S','A',';',
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF ,'<'};
        // for four electrodes together // selected channels

char[] cmdSAc = {'>','S','A',';',
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF ,'<'};
char[] Tamp ={AMP,AMP,AMP,AMP,AMP,AMP,AMP,AMP,
                    AMP,AMP,AMP,AMP,AMP,AMP,AMP,AMP,
                    AMP,AMP,AMP,AMP,AMP,AMP,AMP,AMP,
                    AMP,AMP,AMP,AMP,AMP,AMP,AMP,AMP};
char[] cmdSF = ">SF;HL<".ToCharArray();
char[] cmdSW = ">SW;HL<".ToCharArray();

public Form3()
{

    CMD_SD[4] = (char)0;
    CMD_SD[5] = (char)0;
    CMD_SD[6] = (char)0x13;
    CMD_SD[7] = (char)0x88;
//
    cmdSF[4] = (char)0;
    cmdSF[5] = (char)50;
//
    cmdSW[4] = (char)0x01;
    cmdSW[5] = (char)0x00;
//
    //char cmdC2[21]   = ">Cf;xxxxxxxxxxxxxxx<"; // electrode2
//
    InitializeComponent();
}
```

```
private void btnSerial_Click(object sender, EventArgs e)
{
    sp = new SerialPort(cboPorts.SelectedValue.ToString(),
    115200, Parity.None, 8, StopBits.One);
//
        sp.DataReceived += SerialDataReceived;

    if (!sp.IsOpen)
        try
        {
            sp.Open();
            btnSerial.BackColor = Color.Green;
        }
        catch (Exception)
        {
            btnSerial.BackColor = Color.Red;
            MessageBox.Show("Error opening port");
        }
    else MessageBox.Show(" POrt already open !!");
    // btnSerial.BackColor = Color.Green;

}
delegate void StringArgReturningVoidDelegate(string text, Color color);

private void SerialDataReceived(object sender,
SerialDataReceivedEventArgs e)
{
    SerialPort sp = (SerialPort)sender;
    string s = sp.ReadExisting(); ;
    updateGridSetText(s, Color.Green);
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
}

private void updateGridSetText(string text,Color color = new Color())
{
    if (this.listView1.InvokeRequired)
    {
        StringArgReturningVoidDelegate d;
        d= new StringArgReturningVoidDelegate(updateGridSetText);
        this.Invoke(d, new object[] { text, color });
    }
    else
    {
        if (text.EndsWith("<"))
            text = text + "\r\n";
        ListViewItem lvitem =  this.listView1.Items.Add(text);
        lvitem.ForeColor = color;
        listView1.Scrollable = true;

        // add output to file
    }
}

private void Form3_Load(object sender, EventArgs e)
{
    cboPorts.DataSource = SerialPort.GetPortNames();

    DataGridViewTextBoxCell txt = new DataGridViewTextBoxCell();
```

```
    for (int i = 0; i < 16; i++)
    {
        dgvChannels.Rows.Add(txt);
        dgvChannels.Rows[i].Cells[0].Value = i+1;
        dgvChannels.Rows[i].Cells[1].Value = false;
        dgvChannels.Rows[i].Cells[2].Value = 0;

    }

    dgvChannels.AllowUserToDeleteRows = false;
   dgvChannels.AllowUserToAddRows = false;


}

private void btnFREQ_Click(object sender, EventArgs e)
{
     writeToSP(cmdSF);
 }

private void writeToSP(char[] Command)
{ if (sp == null)
    {
        MessageBox.Show("Port not Open!");
        return;
    }
    if (!sp.IsOpen)
    {
        MessageBox.Show("Port not Open!");
        return;
    }

    sp.Write(Command, 0, Command.Length);
    updateGridSetText(new string(Command )+ "\r\n");

}

private void btnPulsewidth_Click(object sender, EventArgs e)
{

    writeToSP(cmdSW);
}



private void btnCURRENT_Click(object sender, EventArgs e)
{
    char[] cmdc1temp = cmdC1_ZERO;
    char[] cmdc2temp = cmdC2_ZERO;
    //int position = 1;
    //cmdc1temp[4 + position] = (char)30;
    DataGridViewTextBoxCell chkboxCell;

    for (int i = 0; i < 16; i++)
    {
        chkboxCell = (DataGridViewTextBoxCell)(dgvChannels.Rows[i].Cells[2]);
        cmdc1temp[4 + i] = (char)(Int32.Parse(chkboxCell.Value.ToString()));

    }

    writeToSP(cmdc1temp);
    //  System.Threading.Thread.Sleep(3);
```

```
    //   cmdc2temp[4 + 20-16] = (char)30;
    //  writeToSP(cmdc2temp);
      writeToSP(cmdC2_ZERO);

}

private void btnCHANNELS_Click(object sender, EventArgs e)
{
    //for (int j =0; j <cmdSAc.Length -5; j ++)
    //     cmdSAc[4 + j] = (char)255;
    cmdSAc = cmdSA;
    DataGridViewCheckBoxCell chkboxCell;
    for (int i = 0; i < 16; i++)
    {
        chkboxCell = (DataGridViewCheckBoxCell)(dgvChannels.Rows[i].Cells[1]);
        cmdSAc[4 + i] = (char) (((Boolean)chkboxCell.Value? i :255));
        //i = (char)0;
        //cmdSAc[4 + i] = i;
    }
  //  cmdSAc[4+20] = (char)20;

    writeToSP(cmdSAc);
}

private void btnON_Click(object sender, EventArgs e)
{

    writeToSP(SIM_ON);
}

private void btnTRIGGER_Click(object sender, EventArgs e)
{

    writeToSP(SIM_TRIGGER);
}

private void btnOFF_Click(object sender, EventArgs e)
{

    writeToSP(SIM_OFF);
}

private void listView1_SelectedIndexChanged(object sender, EventArgs e)
{

}

private void CLEAR_Click(object sender, EventArgs e)
{
    listView1.Items.Clear();
}

private void button1_Click_1(object sender, EventArgs e)
{
    //writeToSP(txtCommand.Text.Trim().ToCharArray());
    writeToSP(SIM_ON);
  //   writeToSP(cmdSF);

    char[] packet = new char[111];
    for (int j = 0; j < 69; j++)
        packet[j] = cmdSAc[j];

    for (int j = 0; j < 21; j++)
```

```
        packet[j + 69] = cmdC1_ZERO[j];
    for (int j = 0; j < 21; j++)
        packet[j + 69 + 21] = cmdC2_ZERO[j];
//
    writeToSP(packet);
}

private void btnRESET_Click(object sender, EventArgs e)
{
    writeToSP(cmdSA); // reset active channels
    System.Threading.Thread.Sleep(3);
    /// FIRST TESTS WITH RESET C1, C2, SA
    writeToSP(cmdC1_ZERO);
   //System.Threading.Thread.Sleep(3);
    writeToSP(cmdC2_ZERO);
 //
    //    writeToSP(cmdSA); // reset active channels
}

private void dgvChannels_CellValueChanged(object sender, DataGridViewCellEventArgs e)
{
    DataGridView sendergrid = (DataGridView)sender;
    if (sendergrid.Rows.Count <= 0)
        return;
    if (sendergrid.SelectedCells[0].ColumnIndex == 1) // the checkbox
    {
        DataGridViewCheckBoxCell chkboxCell = (DataGridViewCheckBoxCell)(sendergrid.Rows[sendergrid.SelectedC

        if ((Boolean)chkboxCell.Value == true)
        {
            if (sendergrid.Rows[sendergrid.SelectedCells[0].RowIndex].Cells[2].Value.ToString() == "0")
                sendergrid.Rows[sendergrid.SelectedCells[0].RowIndex].Cells[2].Value = txtDefaultAmps.Text;
        }
        else
            sendergrid.Rows[sendergrid.SelectedCells[0].RowIndex].Cells[2].Value = 0;

    }

}
}
}


//////////////////////
//FORM
///////////////
using System;
using System.Drawing;
using System.Windows.Forms;
using System.IO.Ports;
using System.Collections;
using System.IO;
using System.Net.Sockets;

namespace Devcorp.ColorSpaceSample
{
    public partial class Form2 : Form
    {
        SerialPort spBluetooth = null;
        ArrayList points;
        SerialPort sp;
        char[] rxBuffer;
        StreamWriter fs = new StreamWriter("output.txt", false);
```

```
    StreamWriter fsShape;
    Hashtable shapes = new Hashtable();
    String LastLine = "";
    Socket socket = new Socket(AddressFamily.InterNetwork,SocketType.Dgram, ProtocolType.Udp);
    int [] electrodes = new int [] {11,6,11+16,6+16,
10,7,1,7+16,
9,8,2,8+16,
12,5,12+16,5+16,
13,4,13+16, 4+16,
14,3,14+16,3+16};
    public Form2()
    {
        points = new ArrayList();
        rxBuffer = new char[14];
        InitializeComponent();
       // addShapes();

        // DataGridViewRow dr0 = new DataGridViewRow();

        //DataGridViewButtonCell b = new DataGridViewButtonCell();
        //int rowIndex = dataGridView1.Rows.Add(b);
        //dataGridView1.Rows[rowIndex].Cells[0].Value = "name";
        DataGridViewTextBoxCell c = new DataGridViewTextBoxCell();
        for (int i = 0; i < 13+3; i++)
        {
            dataGridView1.Rows.Add(c);
            for (int j = 0; j < 9+1; j++)
                dataGridView1.Rows[i].Cells[j].Value = "";

        }
        dataGridView1.EndEdit();

        for (int i = 0; i < 6; i++)
        {
            dataGridView2.Rows.Add(c);
            for (int j = 0; j < 4; j++)
                dataGridView2.Rows[i].Cells[j].Value = (electrodes[j+4*i])%16;

        }

        dataGridView2.EndEdit();
    }

    private void SerialDataReceived(object sender, SerialDataReceivedEventArgs e)
    {
        SerialPort sp = (SerialPort)sender;
        string s = sp.ReadLine(); ;
        updateGridSetText(s);
        LastLine = s;

        //    throw new NotImplementedException();
    }

    private void SerialDataReceived2(object sender, SerialDataReceivedEventArgs e)
    {
        SerialPort sp = (SerialPort)sender;
        string s = sp.ReadLine(); ;
        updateGridSetText2(s);
        LastLine = s;

        //    throw new NotImplementedException();
    }
    private void updateGridSetText2(string textData)
```

```
        {

            // InvokeRequired required compares the thread ID of the
            // calling thread to the thread ID of the creating thread.
            // If these threads are different, it returns true.
            if (this.txtOutput.InvokeRequired)
            {
                StringArgReturningVoidDelegate d = new StringArgReturningVoidDelegate(updateGridSetText);
                this.Invoke(d, new object[] { textData });
            }
            else
            {
                this.txtOutput.AppendText(textData);
                // add output to file

            }

            char[] data = textData.ToCharArray();
            int x, y;
            for (int i = 0; i < data.Length -1; i ++)
            {   if (data[i] == 1)
                {
                    x = (i % 4) * 2 + 1;
                    y = (i / 4) * 2 + 1;
                    Coord xy = new Coord(x,y,0);
                    updateUI(xy, false);

                        }


            }


        }


        delegate void StringArgReturningVoidDelegate(string text);
        delegate void CoordArgReturningVoidDelegate(Coord xy, Boolean clear);
        delegate void VoidReturningVoidDelegate();
        private void updateGridSetText(string text)
        {

            // InvokeRequired required compares the thread ID of the
            // calling thread to the thread ID of the creating thread.
            // If these threads are different, it returns true.
            if (this.txtOutput.InvokeRequired)
            {
                StringArgReturningVoidDelegate d = new StringArgReturningVoidDelegate(updateGridSetText);
                this.Invoke(d, new object[] { text });
            }
            else
            {
                this.txtOutput.AppendText(text);
                // add output to file



            }
// String format:
//"X:%d;Y:%d;%d;%c;%c;%c;%c;%c;%c;%c;%c;%c;%c;%c;%c;"
//,getX(position),getY(position),position,RxBuffer[4],RxBuffer[6],RxBuffer[8],RxBuffer[10],
//RxBuffer[12],RxBuffer[14],RxBuffer[16],RxBuffer[18],RxBuffer[20],RxBuffer[22],RxBuffer[24],RxBuffer[26])
            if (text.StartsWith("X:") && text.Contains("Y:"))
```

```
        {
            Coord xy = parseXYString(text);

            //      byte capacitance  = getCapacitance(xy);
            updateUI(xy, false);
            if ((xy.x != -1) && (xy.y != -1))
            {
                points.Add(xy);

                String[] timedata = text.Split(',');

                if (timedata.Length > 1)
                {
                    //  fs.Write((xy.x + (xy.y - 1) * 9) + " " + text.Split(',')[1]);
                    fs.Write((xy.x + (xy.y - 1) * 9) +"\n");
                    fs.Flush();
                }
            }
        }
        else

        {
            updateUIMulti();
            // TODO here we should implement another method to show the capacitance
        }
    }


    private byte getCapacitance(Coord xy)
    {
        int capacitanceX = 0;
        int capacitanceY = 0;
        // get electrode number for X
        // get electrode number for y
        switch (xy.x)
        { // the rxbuffer has at index 0 the touch register value
            case 1:
                capacitanceX = rxBuffer[1];
                break;

            case 2:
                capacitanceX = (rxBuffer[1] + rxBuffer[2]) / 2;
                break;

            case 3:
                capacitanceX = rxBuffer[2];
                break;

            case 4:
                capacitanceX = (rxBuffer[2] + rxBuffer[3]) / 2;
                break;

            case 5:
                capacitanceX = rxBuffer[3];
                break;

            case 6:
                capacitanceX = (rxBuffer[3] + rxBuffer[4]) / 2;
                break;

            case 7:
                capacitanceX = rxBuffer[4];
                break;
```

```
            case 8:
                capacitanceX = (rxBuffer[4] + rxBuffer[5]) / 2;
                break;

            case 9:
                capacitanceX = rxBuffer[5];
                break;

        }

        return 255;
    }

    private void updateUIMulti()
    {
        /* Here we will read xy from the touch register directly
         we will make the rows columns calculation
         the result should be a multipoint xy
         draw each point without clear.
         A clear should appear on new data
         TODO next ,make the clear on the release trigger
        */
        //  throw new NotImplementedException();
    }
    private void clearUI()
    {

        if (this.dataGridView1.InvokeRequired)
        {
            VoidReturningVoidDelegate d = new VoidReturningVoidDelegate(clearUI);
            this.Invoke(d, new object[] { });
        }
        else
        {
            for (int j = 0; j < 9 +1; j++) // columns
                for (int i = 0; i < 13 +3; i++) // rows
                    dataGridView1.Rows[i].Cells[j].Style.BackColor = Color.White;

            for (int j = 0; j < 4; j++) // columns
                for (int i = 0; i < 6; i++) // rows
                    dataGridView2.Rows[i].Cells[j].Style.BackColor = Color.White;

        }
        points.Clear();

    }
    private void updateUI(Coord xy, Boolean clear)
    { // this implementation does not provide multi_touch
        /// it returns single X and Y , for each Touch

        int xs; // x for stimulator grid
        int ys; // y for for stimulator grid


        // InvokeRequired required compares the thread ID of the
        // calling thread to the thread ID of the creating thread.
        // If these threads are different, it returns true.
        if (this.dataGridView1.InvokeRequired)
        {
            CoordArgReturningVoidDelegate d = new CoordArgReturningVoidDelegate(updateUI);
            this.Invoke(d, new object[] { xy, clear });
        }
```

```
if (clear)
{
    //for (int j = 0; j < 9; j++) // columns
    //    for (int i = 0; i < 13; i++) // rows
    //        dataGridView1.Rows[i].Cells[j].Style.BackColor = Color.White;
    clearUI();
}
if ((xy.x == 0) || (xy.y == 0)) return;
else
{

    DataGridViewTextBoxCell c;

    /// end clear
    ///
    // end if
    if (xy.x == -1)
    {
        //for (int j = 0; j < 9; j++) // columns
        //{
        //    c = (DataGridViewTextBoxCell)dataGridView1.Rows[xy.y - 1].Cells[j];
        //    //  Color color = Color.FromArgb(new Random().Next() * 255 * 255 * 255 * 255);

        //    c.Style.BackColor = Color.Red;
        //  //  c.Style.BackColor = color;
        //    //c.Style.BackColor = Color.FromArgb((i * 100 + j * 10) % 256, (i * 10 + j * 100
        //    dataGridView1.EndEdit();
        //}
        return;
    }

    if (xy.y == -1)

    {
        //for (int i = 0; i < 13; i++) // rows
        //{
        //    c = (DataGridViewTextBoxCell)dataGridView1.Rows[i].Cells[xy.x - 1];
        //    c.Style.BackColor = Color.Red;
        //    dataGridView1.EndEdit();
        //}
        return;
    }
    else

    {
        c = (DataGridViewTextBoxCell)dataGridView1.Rows[xy.y - 1].Cells[xy.x - 1];
        c.Style.BackColor = Color.Red;
        dataGridView1.EndEdit();
        // map to xy stimulator
        // map color xy stimulator

        xs = (int)(Math.Floor((xy.x -1  )/ 2.5));
        ys = (int)(Math.Floor((xy.y -1 )/ 2.66));
        c = (DataGridViewTextBoxCell)dataGridView2.Rows[ys].Cells[xs];
        c.Style.BackColor = Color.Green;
        dataGridView2.EndEdit();

        return;

    }

    //do the rest for updating the value
}
```

```
}

private struct Coord

{
    public int x, y;
    byte intensity;

    public Coord(int p1, int p2, byte intens)
    {
        x = p1;
        y = p2;
        intensity = intens;
    }

}
private Coord parseXYString(String data)
{
    string X = "0";
    string Y = "0";
    //  int datapos = 0;
    Coord xy = new Coord(0, 0, 255);
    // if (data.StartsWith("X:")&& data.Contains("Y:"))
    //   {
    X = ((data.Substring(2)).Split(':', 'Y', ';'))[0];
    Y = data.Substring(data.IndexOf('Y') + 2).Split('\r', '\n', ';')[0];
    string YValueAndData = data.Substring(data.IndexOf('Y') + 2);
    string[] RemainingData = YValueAndData.Substring(YValueAndData.IndexOf(';') + 1).Split(';'); ///

    rxBuffer = string.Join("", RemainingData).ToCharArray();

    //   }
    Int32.TryParse(X, out xy.x);
    Int32.TryParse(Y, out xy.y);
    return xy;

}

private Coord parseHexString(String data)
{
    Coord xy = new Coord();

    return xy;

}
private void button1_Click(object sender, EventArgs e)
{

    //dataGridView1.Rows[0].Cells[0].Value = new Button();
    DataGridViewTextBoxCell c;
    for (int i = 0; i < 13+3; i++)
    {

        for (int j = 0; j < 8+1; j++)
        {
            c = (DataGridViewTextBoxCell)dataGridView1.Rows[i].Cells[j];
            //  Color color = Color.FromArgb(new Random().Next() * 255 * 255 * 255 * 255);


            c.Style.BackColor = Color.FromArgb((i * 100 + j * 10) % 256, (i * 10 + j * 100) % 256, (20
        }
```

```
        }
        dataGridView1.EndEdit();

    }

    private void btnSerial_Click(object sender, EventArgs e)
    {
        sp = new SerialPort(cboPorts.SelectedValue.ToString(), 115200, Parity.None, 8, StopBits.One);
        if (!chkpacketNew.Checked)
        {
            sp.DataReceived += SerialDataReceived;
            sp.DataReceived -= SerialDataReceived2;
        }
        else
        {
        sp.DataReceived += SerialDataReceived2;
         sp.DataReceived -= SerialDataReceived;
        }
        if (!sp.IsOpen)
            try
            {
                sp.Open();
                btnSerial.BackColor = Color.Green;
            }
            catch (Exception)
            {
                btnSerial.BackColor = Color.Red;
                MessageBox.Show("Error opening port");
            }
        else MessageBox.Show(" POrt already open !!");
        // btnSerial.BackColor = Color.Green;

    }


    private int bitRead(int number, int position)
    {
        // we assume it is 16 bit position
        //return ((number << (16 - position )) >> 16) ;
        return ((number & (0x0001 << position)) >> position);

    }

    private void button2_Click(object sender, EventArgs e)
    {
        fsShape = new StreamWriter("shape.txt", false);
        if ((LastLine.Split(';', ',')).Length < 3)

        {
            textBox2.Text = "Shape Not Found";
            fsShape.Write("0");
            fsShape.Close();
            return;

        }
        int data =  Int32.Parse((LastLine.Split(';', ','))[2]);
        textBox2.Text = (LastLine.Split(';', ','))[2];
        if (!shapes.ContainsKey(data))
        {
            textBox2.Text = "Shape Not Found";
            fsShape.Write("0");
            fsShape.Close();
        }
```

```
    else
    { string shapeCode;
        textBox2.Text = shapes[data].ToString();
        if (textBox2.Text.Contains("ircle"))
            shapeCode = "1";
        else
            shapeCode = "2";
        fsShape.Write(shapeCode);
        fsShape.Close ();
                    }
    //  int data= Int32.Parse(textBox2.Text);
    // int newData = data;
    //  int i=0;
    //  Coord pt = new Coord(0, 0, 255);
    //  updateUI(pt, true);
    //while ((newData != 0) && (i <255))
    //{
    //    pt = new Coord(0, 0, 255);
    //    i++;
    //    pt.x=TouchPadHelper.getX(data, out newData);
    //    txtOutput.AppendText(newData.ToString() + "\r\n ");
    //    pt.y =TouchPadHelper.getY(newData, out newData);
    //  txtOutput.AppendText(newData.ToString() + "\r\n ");
    //    txtOutput.AppendText("X:" + pt.x  + "Y:"  + pt.y + "\r\n ");
    //     if (pt.x > 0 && pt.y > 0)
    //        updateUI(pt, false);
    //}
    //i = 0;
    //while ((newData != 0) && (i < 255))
    //{
    //    pt = new Coord(0, 0, 255);
    //    i++;
    //    pt.x = TouchPadHelper.getX(newData, out newData);
    //    txtOutput.AppendText(newData.ToString() + "\r\n ");
    //    pt.y = TouchPadHelper.getY(data, out newData);
    //    txtOutput.AppendText(newData.ToString() + "\r\n ");
    //    txtOutput.AppendText("X:" + pt.x + "Y:" + pt.y + "\r\n ");
    //    if (pt.x > 0 && pt.y > 0)
    //        updateUI(pt, false);
    //}


}

private bool onlyOneLeft()
{
    throw new NotImplementedException();
}

private void btnClear_Click(object sender, EventArgs e)
{
    clearUI();
    txtOutput.Clear();
    fs.Close();
    fs = new StreamWriter("output.txt", false);
}

private void Form2_Load(object sender, EventArgs e)
{

    dataGridView1.Rows[0].Cells[0].Selected = false;
    dataGridView2.Rows[0].Cells[0].Selected = false;
    cboPorts.DataSource = SerialPort.GetPortNames();
```

```
}

private void button3_Click(object sender, EventArgs e)
{
    if (points.Count > 2)
    {
        Coord pt1 = (Coord)points[0];
        Coord pt2 = (Coord)points[points.Count - 1];
        if ((Math.Abs(pt1.x - pt2.x) <= 1) && (Math.Abs(pt1.y - pt2.y) > 2))
        {
            txtOutput.AppendText("\r\n Vertical line");
            // slope can be infinite

        }
        else
        if ((Math.Abs(pt1.y - pt2.y) <= 1)  && (Math.Abs(pt1.x - pt2.x) > 2))
        {

            txtOutput.AppendText("\r\n Horizontal line");
            //      return; we can get the slope
        }

        else
        {
            double slope = getSlope();
            txtOutput.AppendText("\r\n");
            txtOutput.AppendText("LSM slope =" + slope.ToString("#.##" ));
            txtOutput.AppendText("\r\n");
            txtOutput.AppendText("First-Last slope=" + (pt2.y - pt1.y) / (pt2.x - pt1.x));
        }

    }
    else
    {
        txtOutput.AppendText("Points < 2 ");

    }
    clearUI();
    points.Clear();
}

private double getSlope()
{
    Coord pt1 = (Coord)points[0];
    Coord pt2 = (Coord)points[points.Count - 1];
    //if (false)
     //   return (pt2.y - pt1.y) / (pt2.x - pt1.x);
    //else
    {

        // another method / least square method

        double Ymean = 0;
        double Xmean = 0;
        double Deltaxy=0;
        double Deltaxx=0;
        for (int i = 0; i < points.Count; i++)
        {
            Xmean += ((Coord)points[i]).x;
            Ymean += ((Coord)points[i]).y;
        }
        Xmean = Xmean / points.Count;
        Ymean = Ymean / points.Count;
```

```
            for (int i = 0; i < points.Count; i++)
            {
                Deltaxy += (((Coord)points[i]).x - Xmean)* (((Coord)points[i]).y - Ymean);
                Deltaxx += Math.Pow((((Coord)points[i]).x - Xmean),2) ;
            }
            return (Deltaxy / Deltaxx);
            // we may need filtering of the redundant points
            // maybe by adding a key value hashtable

        }




    }

    private void button4_Click(object sender, EventArgs e)
    {
        char AMP = (char)12;


        char [] SIM_ON = ">ON<".ToCharArray(); // >T< trigger
        char [] SIM_OFF = ">OFF<".ToCharArray();
        char [] CMD_SD = ">SD;xxxx<".ToCharArray();

        char[] cmdC1 = {'>','C',(char)1,';',
AMP,AMP,AMP,AMP,
AMP,AMP,AMP,AMP,
AMP,AMP,AMP,AMP,
AMP,AMP,AMP,AMP,
'<'}; // electode1

        //char cmdC2[21]   = ">Cf;xxxxxxxxxxxxxxxx<"; // electrode2
        char [] cmdC2 = {'>','C',(char)2,';',
AMP,AMP,AMP,AMP,
AMP,AMP,AMP,AMP,
AMP,AMP,AMP,AMP,
AMP,AMP,AMP,AMP,
'<'}; // electode2

        char AMP_Z = (char)0;
        char[] cmdC1_ZERO = {'>','C',(char)1,';',
AMP_Z,AMP_Z,AMP_Z,AMP_Z,
AMP_Z,AMP_Z,AMP_Z,AMP_Z,
AMP_Z,AMP_Z,AMP_Z,AMP_Z,
AMP_Z,AMP_Z,AMP_Z,AMP_Z,
'<'};
        char[] cmdC2_ZERO = {'>','C',(char)2,';',
AMP_Z,AMP_Z,AMP_Z,AMP_Z,
AMP_Z,AMP_Z,AMP_Z,AMP_Z,
AMP_Z,AMP_Z,AMP_Z,AMP_Z,
AMP_Z,AMP_Z,AMP_Z,AMP_Z,
'<'};

        char [] cmdSA = ">SA;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx<".ToCharArra
        char [] cmdSAc = ">SA;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx<".ToCharAr
        char[] Tamp ={AMP,AMP,AMP,AMP,AMP,AMP,AMP,AMP,
            AMP,AMP,AMP,AMP,AMP,AMP,AMP,AMP,
            AMP,AMP,AMP,AMP,AMP,AMP,AMP,AMP,
            AMP,AMP,AMP,AMP,AMP,AMP,AMP,AMP};
        char i = (char)1;  // zero based // active channel
```

```
cmdSAc[4 + i] = i;

CMD_SD[4] = (char)0;
CMD_SD[5] = (char)0;
CMD_SD[6] = (char)0x13;
CMD_SD[7] = (char)0x88;
char [] cmdSF = ">SF;HL<".ToCharArray();
cmdSF[4] = (char)0;
cmdSF[5] = (char)50;

char[] cmdSW = ">SW;HL<".ToCharArray();
cmdSW[4] = (char)0x01;
cmdSW[5] = (char)0x00;

if (spBluetooth == null)
{
    spBluetooth = new SerialPort(cboPorts.SelectedValue.ToString(), 115200, Parity.None, 8, St
   // spBluetooth.DataReceived += SerialDataReceivedBT;
}
if (!spBluetooth.IsOpen)
    spBluetooth.Open();
String s = (String)spBluetooth.ReadExisting();
//  spBluetooth.Write(SIM_OFF, 0, SIM_OFF.Length);
// s = (String)spBluetooth.ReadExisting();
//spBluetooth.Write(Tamp, 0, Tamp.Length);
//  System.Threading.Thread.Sleep(3);
//   spBluetooth.Write(cmdSA, 0, cmdSA.Length);
//    System.Threading.Thread.Sleep(10);
//   s = (String)spBluetooth.ReadExisting();
// aLL CHANNELS OFF


//  spBluetooth.Write(cmdC1_ZERO, 0, cmdC1_ZERO.Length);
//   System.Threading.Thread.Sleep(10);
//    s = (String)spBluetooth.ReadExisting();
//    spBluetooth.Write(cmdC2_ZERO, 0, cmdC2_ZERO.Length);
//    System.Threading.Thread.Sleep(10);
//s = (String)spBluetooth.ReadExisting();
//System.Threading.Thread.Sleep(10);
//spBluetooth.Write(cmdSW, 0, cmdSW.Length);
//s = (String)spBluetooth.ReadExisting();
//System.Threading.Thread.Sleep(10);
//spBluetooth.Write(cmdSF, 0, cmdSF.Length);
//s = (String)spBluetooth.ReadExisting();

//// run

//  spBluetooth.Write(SIM_ON, 0, SIM_ON.Length);
//s = (String)spBluetooth.ReadExisting();
char [] packet = new char [111];
for (int j = 0; j < 69; j++)
    packet[j] = cmdSAc[j];

for (int j = 0; j< 21; j++)
    packet[j + 69] = cmdC1[j];
for (int j = 0; j < 21; j++)
    packet[j + 69 + 21] = cmdC2[j];

///// this block was replaced by below
//spBluetooth.Write(cmdSAc, 0,cmdSAc.Length);
//s = (String)spBluetooth.ReadExisting();
//if (!s.Contains("OK"))
//    MessageBox.Show("errror  +" + s);
```

```
            //spBluetooth.Write(cmdC1, 0,cmdC1.Length);
            //s = (String)spBluetooth.ReadExisting();
            //if (!s.Contains("OK"))
            //    MessageBox.Show("errror  +" + s);

            //spBluetooth.Write(cmdC2, 0,cmdC2.Length);
            //s = (String)spBluetooth.ReadExisting();
            //if (!s.Contains("OK"))
            //    MessageBox.Show("errror  +" + s);
            ///// this block was replaced by below

            spBluetooth.Write(packet, 0, 111);
            s = (String)spBluetooth.ReadExisting();
            if (!s.Contains("OK"))
                MessageBox.Show("errror  +" + s);

            spBluetooth.Write(SIM_ON, 0, SIM_ON.Length);
            s = (String)spBluetooth.ReadExisting();
            if (!s.Contains("OK"))
                MessageBox.Show("errror  +" + s);

            ////////////////////
              spBluetooth.Write(SIM_OFF, 0, SIM_OFF.Length);
            s = (String)spBluetooth.ReadExisting();
            //   spBluetooth.Write(cmdSA, 0, cmdSA.Length);

            //  spBluetooth.Write(SIM_OFF , 0, SIM_OFF.Length);
            // spBluetooth.Write(CMD_SD, 0, CMD_SD.Length);

        }

        private void chkpacketNew_CheckedChanged(object sender, EventArgs e)
        {
            if (sp != null)
                if (chkpacketNew.Checked)

            {
                sp.DataReceived -= SerialDataReceived;
                sp.DataReceived += SerialDataReceived2;
            }
            else
            {
                sp.DataReceived += SerialDataReceived;
                sp.DataReceived -= SerialDataReceived2;
            }

        }

    private void SerialDataReceivedBT(object sender, SerialDataReceivedEventArgs e)
    {
        SerialPort sp = (SerialPort)sender;
        object s = sp.ReadExisting();
        if (this.txtOutput.InvokeRequired)
        {
            StringArgReturningVoidDelegate d = new StringArgReturningVoidDelegate(updateGridSetText);
            this.Invoke(d, new object[] { s });
        }
        else
        {
            this.txtOutput.AppendText((string)s);
            // add output to file
```

```
        }

        //    throw new NotImplementedException();
    }

        private void btnTestForm_Click(object sender, EventArgs e)
        {
            Form3 frm3 = new Form3();
            frm3.ShowDialog();

        }

        private void cboPorts_MouseDoubleClick(object sender, MouseEventArgs e)
        {
            cboPorts.DataSource = SerialPort.GetPortNames();
        }
    }

}
```

B.2 Arduino Code

```
/*
  Snowforce.ino
  Get force data from onboard matrix controller and send it to PC.
  Copyright (c) 2014-2016 Kitronyx http://www.kitronyx.com
  GPL V3.0
  This code is made for BIOCAS 2018 Demo.
  Author: Mohamad Alameh
  COSMIC LAB / DITEN / UNIGE
  mohamad.alameh@edu.unige.it
  alameh.mhd@gmail.com

*/

#include <Wire.h>
#include <Snowforce.h>

Snowforce snowforce;
char AMP = 0;
byte data[160] = {0, };
char    electrodes [24] = {11, 6, 11 + 16, 6 + 16,
                           10, 7, 1, 7 + 16,
                           9, 8, 2, 8 + 16,
                           12, 5, 12 + 16, 5 + 16,
                           13, 4, 13 + 16, 4 + 16,
                           14, 3, 14 + 16, 3 + 16
                          };
char SIM_ON[]    = ">ON<";
char SIM_TRIGGER [] = ">T<";
char SIM_OFF[]   = ">OFF<";
char CMD_SD [] = ">SD;xxxx<";  // delay after trigger

char cmdSF[] = ">SF;HL<";

char cmdSW[] = ">SW;HL<";
```

```
char packet [111] ;
boolean flag = true;
//char cmdC1[21]   = ">Cf;xxxxxxxxxxxxxxx<"; // electode1
char cmdC1[]   = {'>', 'C', 1, ';',
                  AMP, AMP, AMP, AMP,
                  AMP, AMP, AMP, AMP,
                  AMP, AMP, AMP, AMP,
                  AMP, AMP, AMP, AMP,
                  '<'
                 }; // electode1

//char cmdC2[21]   = ">Cf;xxxxxxxxxxxxxxx<"; // electrode2
char cmdC2[]   = {'>', 'C', 2, ';',
                  AMP, AMP, AMP, AMP,
                  AMP, AMP, AMP, AMP,
                  AMP, AMP, AMP, AMP,
                  AMP, AMP, AMP, AMP,
                  '<'
                 }; // electode1
////////////////////////////////////////////

char cmdC1ZERO[]   = {'>', 'C', 1, ';',
                      0, 0, 0, 0,
                      0, 0, 0, 0,
                      0, 0, 0, 0,
                      0, 0, 0, 0,
                      '<'
                     }; // electode1

//char cmdC2[21]   = ">Cf;xxxxxxxxxxxxxxx<"; // electrode2
char cmdC2ZERO[]   = {'>', 'C', 2, ';',
                      0, 0, 0, 0,
                      0, 0, 0, 0,
                      0, 0, 0, 0,
                      0, 0, 0, 0,
                      '<'
                     }; // electode1
////////////////////////////
// sa: 69
char cmdSA[]  =
{'>','S','A',';',
255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,
'<'};
// for four electrodes together // selected channels
char cmdSAc[] ={'>','S','A',';',
255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,
'<'};
char Tamp[]  = {AMP, AMP, AMP, AMP, AMP, AMP, AMP, AMP,
```

```
                AMP, AMP, AMP, AMP, AMP, AMP, AMP, AMP,
                AMP, AMP, AMP, AMP, AMP, AMP, AMP, AMP,
                AMP, AMP, AMP, AMP, AMP, AMP, AMP, AMP
               };

void setup()
{
//
  cmdSF[4] = 0;
  cmdSF[5] = 50;
//
  cmdSW[4] = 0;
  cmdSW[5] = 200;
//
  CMD_SD [4] = 0 ;
  CMD_SD [5] = 0;
  CMD_SD [6] = 0;
  CMD_SD [7] = 20;
  Wire.begin(); // start communication with the onboard force controller
  Serial.begin(115200);  // start serial for output
  //  delay (5000); // leave time for bluetooth

  Serial1.begin(115200);
  snowforce.begin(); // start tactile sensing part of snowboard
  //Serial.print("A");
  delay(5000);
//
//  Serial1.print (SIM_ON);
//
  Serial1.write (cmdC1ZERO, 21);
  Serial1.write (cmdC2ZERO, 21);
//  Serial1.write (cmdSW, 7);
//  Serial1.write (cmdSF, 7);

  Serial1.write (cmdSA, 69);
  while (Serial1.available() > 0)
    Serial1.read();

  // Serial1.write (cmdC1,21);
  //  Serial1.write (cmdC2,21);
  //        Serial1.write (SIM_OFF,5);
  //        delay(2);
  Serial1.write (SIM_ON,4);
Serial.print("sending commands");
 // to be removed:
 cmdC1[4+2] = 30;
 cmdC1[4+1] = 30;
 //cmdC2[4+20-16] = 30;
 cmdSAc[4+2] = 2;
 cmdSAc[4+1] = 1;
// cmdSAc[4+20] = 20;
 Serial1.write (cmdSAc,69);
 Serial1.write (cmdC1,21);
 Serial1.write (cmdC2,21);
  delay(1000);
 while (Serial1.available() >0)
  Serial.print(Serial1.readString());

}

void loop()
{
  snowforce.read(data);
```

```
  // pressure mapping data
  // tactile sensing part always give maximum
  // number of data (10x16 = 160)
  //cmdSAc  = cmdSA ; // todo
  //Serial1.print ("A");
  //Wire.write("A");
  // send data only pc wants it
  delay (50);
  // if (Serial.available() > 0)
  // {
  int inByte = Serial.read();
  //  Serial.print(inByte);

  //  snowforce.read(data);

  for (int i = 0; i < 160; i++)
  {
    if (data[i] > 1 )
    {
      int x = i % 10 + 1;
      int  y =  (i / 10) + 1;
      Serial.print("X:");//a
      Serial.print (x);//a
      Serial.print(";");//a
      Serial.print ("Y:"); //a
      Serial.print (y);//a
      //           // Serial.print(data[i]);
      Serial.print(";");//a
      int xs = (int)(floor((x - 1  ) / 2.5));
      int ys = (int)(floor((y - 1 ) / 2.66));
      //  Serial.print(xs);
      // Serial.print(";");
      // Serial.print(ys);
      Serial.print("\r\n");//a

      // Serial.write("A");
      // xs and xy's are the position on the stimulator (0 index based)
      stimulate(xs, ys);
    }

  }

  resetBuffers();
  //    }
  //   Serial.println(data[159]);
  //  digitalWrite(LED_BUILTIN, LOW);
}
void stimulate(int xs, int ys) {


  char realElectrodeNum = electrodes [xs + (ys * 4)] ;
  int el = 4 + realElectrodeNum - 1 ;
  cmdSAc [4 + realElectrodeNum - 1] = realElectrodeNum - 1 ;
  if (el <16)
    cmdC1 [el] = 30 ;
  if (el >=16)
    cmdC2 [el%16] = 30 ;

  // todo change also amplitudes
//  if (xs + (ys * 4) >= 24 )
//    Serial1.print("XXXX");
//
  for (int i = 0 ; i < 69; i ++ )
```

```
    packet[i] = cmdSAc[i];
//
  for (int i = 0 ; i < 21; i ++ )
    packet[i + 69] = cmdC1[i];
  for (int i = 0 ; i < 21 ; i ++ )
    packet[i + 69 + 21] = cmdC2[i];
//   Serial1.write (packet, 111);

  if (flag)
    {
   Serial1.write (SIM_ON,4);
   delay(3);
   flag = false;
   Serial.write("On is called");
     }

         Serial1.write (cmdSAc,69);
         delay(1);
          Serial1.write (cmdC1,21);
          delay (1);
          Serial1.write (cmdC2,21);
          delay (1);
       Serial1.write (CMD_SD,9);
          delay(2);

         Serial1.write (SIM_OFF,5);
          delay(1);
                  flag = true;
  Serial.print("\r\n")    ;
  // Serial1.write (SIM_ON,4);
  //  delay(2);
}

void resetBuffers() {
  for (int i = 4; i < 69 - 1; i++) // buffer length - 2
  // because the last one is a terminator
    cmdSAc[i] = cmdSA[i] ;
  for (int i = 4; i < 20; i++)
    cmdC1[i] = cmdC1ZERO[i] ;
  for (int i = 4; i < 20; i++)
    cmdC2[i] = cmdC2ZERO[i] ;

}
```

Appendix C

Recursive Neural Network Codes

All codes of this appendix can be found on:

https://github.com/alamehm/IEEE_SENSORS_Touch_modality_

C.1 Data Preprocessing

C.1.1 Dataset A

```
%%%Matlab Code
clc
close all
clear all
mainfile = 'C:\input_PATH   \';
Destfile = 'C:\destination';
Parts = dir(mainfile);
Partslist = {Parts.name};
Partlist = Partslist(3:end);
ST = zeros(64,768);
ED = zeros(64,768);
for j=1:3
    Partnum = Partlist{j};
    Partfile = sprintf('%s%s%s',mainfile,'\',Partnum);
    PartDestfile = sprintf('%s%s%s',Destfile,'\',Partnum);
    Files = dir(Partfile);
    filelist = {Files.name};
    filelist = filelist(3:end);
    for z=1:260
        fileN = filelist{z};
        filename = sprintf('%s%s%s',Partfile,'\',fileN);
        filenameDest = sprintf('%s%s%s',PartDestfile,'\',fileN);
        alldata = importdata(filename);
        data = alldata (:,2:end);
        fire = -1;
        for p= 1:30000
```

```
            if fire > 0
                break
            end
            for k = 1:16
                if data(p,k)<1.62 || data(p,k)>1.68
                fire = p;
                break;
                end
            end
        end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if fire > 50
            start = fire - 50;
        elseif fire < 50
            start = 1;
        end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        image=data(start:end,:);
        [ImR,ImC] = size(image);
        ST(j,z) = start;
        ed = -1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for p= ImR:-1:1
            if ed > 0
                break
            end
            for k = 1:16
                if image(p,k)<1.62 || image(p,k)>1.68
                ed = p;
                break;
                end
            end
        end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if start > 23856
          take = 29999;
          start = 23856;
        elseif ed-start < 6143
            take = ed + (6143-(ed-start));
        elseif ed-start > 6143
            take = start+6143;
        end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        image=data(start:take,:);
%       size(image)
        Fig=image;
        finalDest = erase(filenameDest,'.lvm');
        filesave = sprintf('%s%s',finalDest,'.txt');
        writematrix(Fig,filesave);
    end
end
```

C.1.2 Dataset B

```
%%%Matlab Code
clc
close all
mainfile = 'Pathtodataset\Dataset';
Destfile = 'outputpath\Images (RGB)';
```

```
Parts = dir(mainfile);
Partlist = {Parts.name};
Partlist = Partlist(3:end);
for j=1:3
    Partnum = Partlist{j};
    Partfile = sprintf('%s%s%s',mainfile,'\',Partnum);
    PartDestfile = sprintf('%s%s%s',Destfile,'\',Partnum);
    Files = dir(Partfile);
    filelist = {Files.name};
    filelist = filelist(3:end);
    B = zeros(4,4);
    A = zeros(64,64);
    for z=1:260
        fileN = filelist{z};
        filenameDest = sprintf('%s%s%s',PartDestfile,'\',fileN);
        filename = sprintf('%s%s%s',Partfile,'\',fileN);
        data = importdata(filename);
        n=0;x=0;u=0;l=1;o=0;
        B = zeros(4,4);
        A = zeros(64,64);
        for i=1:256
            C = data(i,:);
            for e=1:4
                B(e,:) = C(e*4-3:e*4);
            end
            o=o+1;
            A(l*4-3:l*4,o*4-3:o*4)= B;
            if rem(i,16)==0
                l=l+1;
                o=0;
            end
        end
        finalDest = erase(filenameDest,'.txt');
        filesave = sprintf('%s%s',finalDest,'.bmp');
        Im = uint8(((A-min(min(A)))/(max(max(A))-min(min(A))))*256);
        rgbimage = cat(3, Im, Im, Im);
%         GrayImage = mat2gray(A,[min(min(data)) max(max(data))]);
%         imshow(GrayImage);
        imwrite(rgbimage,filesave);
    end
end
```

C.1.3 Dataset C

```
%%%Matlab Code

clc
close all
clear all
mainfile = 'C:\input_path\'; %% Dataset file
Destfile = 'C:\Destination\'; %% Destination file
Parts = dir(mainfile);
Partslist = {Parts.name};
Partlist = Partslist(3:end);

for j=1:3
    Partnum = Partlist{j};
    Partfile = sprintf('%s%s%s',mainfile,'\',Partnum);
    PartDestfile = sprintf('%s%s%s',Destfile,'\',Partnum);
```

```
    Files = dir(Partfile);
    filelist = {Files.name};
    filelist = filelist(3:end);
    for z=1:260
        fileN = filelist{z};
        filename = sprintf('%s%s%s',Partfile,'\',fileN);
        filenameDest = sprintf('%s%s%s',PartDestfile,'\',fileN);
        alldata = importdata(filename);
        out = 20;
        slot = 614;
        B = zeros(out,16);
        for i=1:16
            C = alldata(:,i);
            u=1;
            for e=1:out
                if e<out
                   B(e,i) = mean(C(u:slot+u));
                elseif e==out
                   B(e,i) = mean(C(u:end));
                end
            u=u+slot/2;
            end
        end
        filesave = sprintf('%s',filenameDest);
        writematrix(B,filesave)
   end
end
```

C.2 LSTM/GRU Network Training

```
# this functions loads the dataset from the files (python)
def load_dataset5 (fold = 0,prefix=''):
        #print (prefix)


        X1 = read_csv(prefix+'rolling/'+'rolling_all.txt', header=None, delim_whitespace=False)
        X2 = read_csv(prefix+'sliding/'+'sliding_all.txt', header=None, delim_whitespace=False)
        X3 = read_csv(prefix+'brushing/'+'brushing_all.txt', header=None, delim_whitespace=False)
        print (prefix)

        mu = np.average([np.average(X1),np.average(X2),np.average(X3)])
        std = np.std([np.std(X1),np.std(X2),np.std(X3)])
        n_features = 768# tactnet2850 896#8192#2048#
        timelength = 24#20#200

        #A=X1.values
        #B=X2.values
        #Xlen= X1.shape[0]
        X1=X1.values
        X2=X2.values
        X3=X3.values

        X = np.zeros((X1.shape[0]*3,X1.shape[1]))
        X [ 0:X1.shape[0],:] = X1
        X [X1.shape[0]:X1.shape[0]*2,:]=X2
        X [X1.shape[0]*2:X1.shape[0]*3,:]=X3
```

```
        #X = (X-mu)/std ; print (" with regularization");
        print ("without regularization")
        X = np.reshape((X),(X.shape[0]//timelength,-1,n_features))


        Y = read_csv(prefix + 'y.txt', header=None, delim_whitespace=True)
        Y = Y.values
        y= Y
        #y = to_categorical(Y)
        #X,y = shuffle(X,y)
        skf = StratifiedKFold(n_splits=5,shuffle = False,random_state=None)
        #print (skf.get_n_splits(X, y))
        lst = list(skf.split(X,y))
        train_index, test_index  = list(skf.split(X,y))[fold]
        return X[train_index],to_categorical( y[train_index]), X[test_index],
        to_categorical(y[test_index])


# Training and Testing

# fit and evaluate a model
def evaluate_model(trainX, trainy, testX, testy,epochs,batch_size):
#Building and training the LSTM network

        #number of parameters of a single LSTM layer:
        #(4 * ((size_of_input + 1) * size_of_output + size_of_output^2))
        verbose = 0
        n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2], trainy.shape[1]
        #print ("features " , n_features)
        #print ("Outputs " , n_outputs)
        model = Sequential()
#       model.add(LSTM(100, input_shape=(n_timesteps,n_features),return_sequences=True)) ;print ("first lst
        model.add(LSTM(10, input_shape=(n_timesteps,n_features))); print ("LSTM layer 10 neurons") # origi
        #model.add(Dropout(0.5))
        #model.add(Dense(10, activation='relu')) # original 200
        model.add(Dense(n_outputs, activation='softmax'))
        #model.summary()
        model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
        # fit network
        history = model.fit(trainX, trainy,  epochs=epochs, batch_size=batch_size, verbose=verbose)


        print (datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S.%f')[:-3]) # evaluate model
        _, accuracy = model.evaluate(testX, testy, batch_size=batch_size, verbose=0)
        print (testy.shape[0] , "samples")
        print( datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S.%f')[:-3])
        print(strftime("%d/%m/%Y %H:%M:%S +0000", gmtime()))
        print ( "verbose ",verbose, " epochs ",epochs, " batch_size ",batch_size)
        return accuracy



def evaluate_model_GRU(trainX, trainy, testX, testy,epochs,batch_size):
#Building and training the GRU network
        verbose = 0
        n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2], trainy.shape[1]

        model = Sequential()

        model.add(GRU(12, input_shape=(n_timesteps,n_features))); print ("GRU layer 12 neurons") # origina

        model.add(Dense(n_outputs, activation='softmax'))
        #model.summary()
```

```
        model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
        # fit network
        history = model.fit(trainX, trainy,  epochs=epochs, batch_size=batch_size, verbose=verbose)


        # evaluate model

        _, accuracy = model.evaluate(testX, testy, batch_size=batch_size, verbose=0)
        print ( "verbose ",verbose, " epochs ",epochs, " batch_size ",batch_size)
        return accuracy

# summarize scores
def summarize_results(scores):
        print(scores)
        m, s = mean(scores), std(scores)
        print('Accuracy: %.3f%% (+/-%.3f)' % (m, s))

# run an experiment
def run_experiment(myDataset,epochs,batch_size,repeats=3):
        # load data
        trainX, trainy, testX, testy = myDataset
        # repeat experiment
        scores = list()
        for r in range(repeats):
                score = evaluate_model(trainX, trainy, testX, testy,epochs,batch_size)
                #score = evaluate_model(trainX, trainy, testX, testy,epochs,batch_size)
                score = score * 100.0
                #print('>#%d: %.3f' % (r+1, score), strftime("%d/%m/%Y %H:%M:%S +0000", gmtime()))
                scores.append(score)
        # summarize results
        summarize_results(scores)
        print(strftime("%d/%m/%Y %H:%M:%S +0000", gmtime()))

print ("starting ......")
gc.enable()
for  ff in range(0,2):
        myDataset = load_dataset5(fold = ff, prefix = 'path/to/directory')
        print ("starting Fold ", ff )
        for i in [48]: #[48,96]:
                for j in [96]:#[48,96]:

                        print ("Fold ", ff, "epochs:",i,", batch: ",j)
                        run_experiment(myDataset,epochs=i, batch_size=j,repeats = 10)
                        gc.collect()
print ("finished!")
```